

DEEP METRIC LEARNING WITH DISTANCE SENSITIVE ENTANGLED
TRIPLET LOSSES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KAAN KARAMAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2021

Approval of the thesis:

**DEEP METRIC LEARNING WITH DISTANCE SENSITIVE ENTANGLED
TRIPLET LOSSES**

submitted by **KAAN KARAMAN** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalipçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İlkey Ulusoy
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. A. Aydın Alatan
Supervisor, **Electrical and Electronics Engineering, METU**

Examining Committee Members:

Assist. Prof. Dr. Elif Vural
Electrical and Electronics Engineering, METU

Prof. Dr. A. Aydın Alatan
Electrical and Electronics Engineering, METU

Assoc. Prof. Dr. İ. Aykut Erdem
Computer Engineering, Koç University

Assist. Prof. Dr. R. Gökberk Cinbiş
Computer Engineering, METU

Assist. Prof. Dr. Aykut Koç
Electrical and Electronics Engineering, Bilkent University

Date:

12.02.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Kaan Karaman

Signature :

ABSTRACT

DEEP METRIC LEARNING WITH DISTANCE SENSITIVE ENTANGLED TRIPLET LOSSES

Karaman, Kaan

M.S., Department of Electrical and Electronics Engineering

Supervisor : Prof. Dr. A. Aydın Alatan

February 2021, 101 pages

Metric learning aims to define a distance that is able to measure the semantic difference between the instances in a dataset. The most recent approaches in this area mostly utilize deep neural networks as their models to map the input data into a feature space by finding appropriate distance metrics between the features. A number of loss functions are already defined in the literature based on these similarity metrics to discriminate instances in the feature space. In this thesis, we particularly focus on triplet loss functions in order to designate their gradients. It is argued that the gradients of the vanilla triplet loss function do not force the instances in a triplet along the *right* direction with the *right* magnitude. Hence, the similarities between the instances in a triplet and the natural phenomena of a free electrostatic charge being affected by several forces due to the other charged bodies located in certain coordinates in the space are exploited to determine the right direction and magnitude. Considering the partial gradients of the loss function with respect to the anchor, positive and negative instances of any valid triplet generated from the dataset, four novel triplet loss functions are proposed that cope with the problem pointed out. It is shown that these loss

gradients gradually solve the drawbacks of the vanilla loss function. The performance increment of these losses, especially the METU loss, over the other triplet losses is presented by the results of several fair experiments on a commonly used fine-grained dataset: CUB200-2011. The results of the proposed techniques are comparable with respect to the score values of the state-of-the-art methods in the deep metric learning topic.

Keywords: Deep Learning, Metric Learning, Fine-grained Classification, Image Classification, Image Retrieval, Object Classification

ÖZ

DERİN METRİK ÖĞRENME İÇİN UZAKLIĞA DUYARLI DOLAŞIK ÜÇÜZ MALİYET FONKSİYONLARI

Karaman, Kaan

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi : Prof. Dr. A. Aydın Alatan

Şubat 2021 , 101 sayfa

Metrik öğrenmenin amacı veri kümesindeki örnekler arasındaki anlamsal farkı ölçen bir mesafe tanımlamaktır. En son teknoloji yaklaşımlar, girdileri öznitelik uzayına eşleyen model olarak derin sinir ağlarını kullanır. Literatürde, öznitelik uzayındaki örnekleri ayırt etmek için bu benzerlik ölçütlerine dayalı olarak bir dizi kayıp fonksiyonu tanımlanmıştır. Bu çalışmada, gradyan problemini belirlemek için standart üçlü kayıp fonksiyonunu analiz ettik. Daha açık ifade etmek gerekirse, standart üçlü kayıp fonksiyonunun gradyanlarının, üçlüdeki örnekleri doğru büyüklüklerle doğru yönlere itmediğini gözlemledik. Daha sonra, serbest hareket edebilen bir yüklü cismin kuvvetler nedeniyle hareket etmesi ile kayıp fonksiyonunun kısmi gradyanları nedeniyle eniyilenmesi arasındaki benzerliği gösterdik. Bu durumdan yola çıkarak, dört orijinal maliyet fonksiyonu önerdik. Bu dört kayıp fonksiyonu, diğer üçlü kayıp fonksiyonlarının kusurlarını kademeli olarak çözmektedir. Bu kayıpların standart üçlü kayıp fonksiyonuna göre performans artışı, özellikle METU maliyet fonksiyonu için, kendi tasarladığımız adil deneylerin sonuçlarıyla gösterilmiştir. Ayrıca, yaygın olarak kulla-

nılan bir ince-detaylı veri kümesi üzerinde gerçekleştirilen bu deneylerin sonuçlarını da gösterdik. Bu veri kümesi, CUB-200-2011'dir. Ek olarak, bu çatı altında yapılan deneylerde önerilen kayıpların aldığı sonuçlar, derin metrik öğrenme konusundaki aynı kategoride yer alan son teknoloji yöntemlerin performans skor değerleriyle karşılaştırılabilecek seviyededir.

Anahtar Kelimeler: Derin Öğrenme, Metrik Öğrenme, İnce Detaylı Sınıflandırma, Görüntü Sınıflandırma, Görüntüden Geri Kazanım, Nesne Sınıflandırma

To my lovely wife and precious family

ACKNOWLEDGMENTS

First of all, I would like to thank Prof. Dr. A. Aydın Alatan for his great guidance and patience during this thesis. I feel remarkably fortunate to have the opportunity to write this thesis under the supervision of him. His valuable advice during the process of thought, careful analyses about the core ideas proposed in this work, and constructive feedback helped me to improve both myself and this thesis.

I would like to express my appreciation to the members of my thesis committee, Assist. Prof. Dr. Elif Vural, Assoc. Prof. Dr. İ. Aykut Erdem, Assist. Prof. Dr. R. Gökberk Cinbiş and Assist. Prof. Dr. Aykut Koç for their positive attitudes and worthwhile comments.

Moreover, I am quite lucky to work with Dr. Erhan Gündoğdu and Dr. Berkan Solmaz and conduct research with their comprehensive knowledge of various topics. Their moral support and invaluable suggestions facilitate me to overcome several challenges I encountered during this period.

I would like to acknowledge Dr. Veysel Yücesoy for the fruitful discussions and his constructive feedback. Furthermore, I am grateful to him for the excellent research environment he provided. In addition, I would like to express my gratitude to all members of ASELSAN Research Center for a friendly environment, and in particular to Safa Onur Şahin for his great support and friendship.

I would like to acknowledge The Scientific and Technological Research Council of Turkey (TUBITAK) for financial support.

Last but not least, I wish to thank my family for their endless support and patience during my whole life. I know and feel they are right by my side whenever I need them. I would particularly like to thank the love of my life, my wife Cansu Zeytun Karaman, for her infinite support. She, who is also my best friend, has recovered my motivation whenever I was demoralized during this period. Beyond a shadow of a doubt, I would not be able to finalize this work without her deep love.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vii
ACKNOWLEDGMENTS	x
TABLE OF CONTENTS	xi
LIST OF TABLES	xvi
LIST OF FIGURES	xviii
LIST OF ABBREVIATIONS	xxi
CHAPTERS	
1 INTRODUCTION	1
1.1 Emerging of Deep Networks	2
1.2 Deep Metric Learning	3
1.3 Datasets for Image Retrieval	4
1.4 Problem Statement	5
1.5 Motivation	7
1.6 Contributions	7
2 AN OVERVIEW ON DEEP METRIC LEARNING	9

2.1	State-of-the-Art Methods of Deep Metric Learning	9
2.1.1	Loss Functions in Metric Learning	11
2.1.1.1	Classification Losses	12
2.1.1.2	Tuplet-based Losses	13
2.1.1.3	Batch-based Losses	19
2.1.2	Mining Techniques	22
2.1.3	Advanced Methods	25
2.2	Metrics for Performance Analysis	25
2.2.1	Recall at K	27
2.2.2	Precision at K	28
2.2.3	R-Precision	29
2.2.4	Mean Average Precision at K	30
2.3	Benchmark Datasets	30
2.3.1	Definition of Fine-Grained Datasets	31
2.3.2	Challenges of Fine-Grained Datasets	32
2.3.2.1	Sub-Category Challenge	32
2.3.2.2	Localization Problem	33
2.3.2.3	Labeling Difficulties	33
2.3.3	Benchmark Datasets of Deep Metric Learning	34
2.3.4	Different Splitting Techniques of the Datasets	34
3	DISTANCE SENSITIVE ENTANGLED LOSS FUNCTION	37

3.1	A Weak Analogy between Feature Vectors and Point Charges	38
3.1.1	Derivation of the Novel Loss Function	41
3.1.2	Special Case #1: Equilateral Triangle	43
3.1.3	Special Case #2: Non-Equilateral Triangle	45
3.1.4	Losses with Non-Constant Pull and Push Functions	46
3.2	Hyper-Parameters of the Proposed Loss Function	52
3.2.1	Discussion on the Ratio between Pull and Push Forces	53
3.2.2	Discussion on the Power of the Forces	54
3.2.3	Discussion on the m_1 and m_2 for various (s, r, ρ) 's	60
4	EXPERIMENTS AND DISCUSSIONS	61
4.1	Experimental Setup	61
4.1.1	Testing Platform	62
4.1.2	Data Generation and Network Architecture for the Toy Dataset	63
4.2	Experimental Results	63
4.2.1	Experiments on the Toy Dataset	63
4.2.2	Experiments for the Loss Functions in the Literature	65
4.2.3	Experiments for the Proposed Loss Function	70
4.3	Discussions on Experimental Results	75
4.3.1	The Effects of the Losses without any Mining	76
4.3.2	Comparing Best Results for the Methods	78

5	CONCLUSIONS	81
5.1	Summary	81
5.2	Conclusions	82
5.3	Future Work	83
	REFERENCES	85

APPENDICES

A	CLASS STRUCTURE OF OUR FRAMEWORK	91
A.1	General Structure	91
A.2	Public Functions and Properties of the Classes	92
A.2.1	Dataset Class	92
A.2.2	Transform Class	92
A.2.3	Splitter Class	93
A.2.4	Sampler Class	93
A.2.5	Miner Class	94
A.2.6	Network Class	95
A.2.7	Loss Class	95
A.2.8	Trainer Class	96
A.2.9	Log Writer Class	96
A.2.10	Score Class	97
A.2.11	Tester Class	97
A.2.12	Experiment Class	98

B DISCUSSION ON THE FIRST MARGIN AND THE HINGE FUNCTION FOR $R = 0$ CASE 99

LIST OF TABLES

TABLES

Table 2.1	The table of fine-grained benchmark datasets. The number of classes and the total number of images that are included in each dataset are shown in this table, as well as the meta-class of them.	34
Table 3.1	The table of the forces of all the triplet losses found in literature in chronological order (from 2014 to 2015).	49
Table 3.2	The table of the forces of all the triplet losses found in literature in chronological order (from 2015 to 2020).	50
Table 3.3	The table of the forces of the proposed triplet losses.	51
Table 3.4	The table for the hyper-parameters of the experiments we designed.	60
Table 4.1	The train results of the triplet loss function methods in the literature trained and tested in our framework on the <i>toy</i> dataset with various hyper-parameters (denoted as <i>HP</i>).	64
Table 4.2	The test results of the triplet loss function methods in the literature trained and tested in our framework on <i>CUB200-2011</i> dataset [40] with various hyper-parameters (denoted as <i>HP</i>). (Part-1)	65
Table 4.3	The table for the test results of the triplet loss function methods in the literature trained and tested in our framework on <i>CUB200-2011</i> dataset [40] with various hyper-parameters (denoted as <i>HP</i>). (Part-2)	66
Table 4.4	The table for the test results of the angular triplet loss function without hinge function trained and tested in our framework on <i>CUB200-2011</i> dataset [40] with various hyper-parameters (denoted as <i>HP</i>).	67
Table 4.5	The table for the test results of the moving triplet loss function methods in the literature trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>). (Part-1)	68

Table 4.6	The table for the test results of the moving triplet loss function methods in the literature trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>). (Part-2)	69
Table 4.7	The table for the test results of the moving triplet loss function methods in the literature trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>). (Part-3)	70
Table 4.8	The table for the test results of our distance sensitive triplet loss function methods without the effect of the hinge function (which means we selected $m_1 = 40$ and $m_2 = 1000$) trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>).	71
Table 4.9	The table for the test results of the location-aware triplet loss function methods in the literature trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>).	72
Table 4.10	The table for the test results of our distance sensitive triplet loss function methods with hinge loss and $m_1 = -2.5$, trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>).	73
Table 4.11	The table for the test results of our distance sensitive triplet loss function methods with hinge loss, trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>). (Part-1)	74
Table 4.12	The table for the test results of our distance sensitive triplet loss function methods with hinge loss, trained and tested in our framework on <i>CUB200-2011</i> dataset with various hyper-parameters (denoted as <i>HP</i>). (Part-2)	75
Table 4.13	The table for the test results of all the triplet loss functions without hinge loss trained and tested in our framework on <i>CUB200-2011</i> dataset.	77
Table 4.14	The table for the best test results of all the triplet loss functions trained and tested in our framework on <i>CUB200-2011</i> dataset.	78

LIST OF FIGURES

FIGURES

Figure 1.1 The architecture of LeNet. The figure is taken from the original paper [21].	2
Figure 1.2 The architecture of AlexNet. (The figure from the paper [20]) . . .	2
Figure 2.1 The Siamese networks [2] are proposed for solving the problem of determining whether the signatures in the given two images are the same or not. The trainable parameters of the models in this figure are shared (i.e., they are equal) with each other. They extract the features of the images in different time intervals; next, embedding values for these images are compared to each other. The figure is taken from the paper [2].	14
Figure 2.2 Similar to the Siamese networks [2] shown in Figure 2.1, all the three networks share their trainable parameters to each other (figure of the triplet network is taken from [14]).	15
Figure 2.3 Selection the tuplets for lifted structure loss (Figure from [33]). . .	20
Figure 2.4 A mini-batch for N-Pairs loss function. (a), (b), and (c) show a batch for triplet, (N+1)-tuplet and N-pairs losses, respectively (Figure from [31]).	21
Figure 2.5 2-D representation of the embedding space. The margin, m , is a hyper-parameter of the miner. The coordinates of the samples A , P , and N represent the feature vectors of f_a , f_p , and f_n , which are embeddings of anchor, positive, and negative items of a triplet, respectively. Region #1 belongs to the <i>easy samples</i> , which do not create any gradient for margin-based losses, such as ranking loss. Region #2 is used for <i>semi-hard</i> mining. Finally, the items in Region #3 and #4 are accepted as <i>hard samples</i>	24
Figure 2.6 The tree for evaluation scores that are frequently used in fine-grained image classification and retrieval.	26

Figure 2.7	Difference between the fine and coarse-grained datasets (figure inspired by [39]).	32
Figure 2.8	The images at the same column of the figure belong to the same types of aircraft, but the visual similarity is smaller in each row. (Figure is inspired by [39]).	32
Figure 2.9	The train and test sets of different types of splits for the CUB-200-2011 [40] dataset as shown in Figure 2.8.	35
Figure 3.1	In this example illustration, it is assumed that all particles have the same amount of electrical charge. While Objects #1 & #2 have the same sign, and Object #3 has a reverse sign. Hence, the first two attract each other and repel the third one, when $\epsilon_r < 0$. The interaction between particles i and j is shown as the force denoted by F_{ij} . The resultant force exerted on particle i is calculated by the vector summation ($F_i = \sum_{\forall j} F_{ij}$). Each particle moves in the direction of its resultant electrical force vector. The location of the object #1, #2, and #3 are at (x_1, y_1) , (x_2, y_2) , and (x_3, y_3) , respectively.	39
Figure 3.2	The 2-dimensional representation of embedding space in equilateral triangle case. The equality $D_{an} = D_{ap} = D_{pn} = 1$ is hold. The objects A and P are belonging to the same class, whereas the classes of the object N is different from them.	43
Figure 3.3	The 2-dimensional representation of feature space in the non-equilateral triangle case. Note that the inequalities $D_{ap} \neq D_{an}$, $D_{ap} \neq D_{pn}$, $D_{an} \neq D_{pn}$ are hold.	45
Figure 3.4	The 2-dimensional representation of the embedding space when the anchor instance (denoted as A) is at $(-1, 0)$, the positive one (P) is at $(1, 0)$, and the negative (N) is at $(0, \sqrt{3})$. The sub-figures show the different ρ values.	55
Figure 3.5	The 2-dimensional representation of the mined regions by in-loss-miner for the negative items when the anchor instance (denoted as A) is at $(0, 0)$, the positive one (P) is at $(0.5, 0)$. The sub-figures show the different r values. The other hyper-parameters are as follows: $s = 0$, $\rho = 12/7$. The negative items in the yellow region are being affected by force caused by both anchor and positive instances.	56

Figure 3.6	The 2-dimensional representation of the embedding space when the anchor instance (denoted as A) is at (0, 0), the positive one (P) is at (1, 0), and the negative (N) is at (0, 0.5). The sub-figures show the different r values. The other hyper-parameters are as follows: $s = 0, \rho = 12/7$	57
Figure 3.7	The 2-dimensional representation of the embedding space when the anchor instance (denoted as A) is at (0, 0), the positive one (P) is at (0.5, 0), and the negative (N) is at (0, 1). The sub-figures show the different (s, r) values. The hyper-parameter ρ is equal to $12/7$	59
Figure A.1	The figure about the relationship between the class objects in the implemented framework.	91
Figure A.2	The public methods and variables of dataset class objects.	92
Figure A.3	The public methods and variables of splitter class objects.	93
Figure A.4	The public methods and variables of sampler class objects.	93
Figure A.5	The public methods and variables of miner class objects.	94
Figure A.6	The public methods and variables of network class objects.	95
Figure A.7	The public methods and variables of loss class objects.	95
Figure A.8	The public methods and variables of trainer class objects.	96
Figure A.9	The public methods and variables of score class objects.	97
Figure A.10	The public methods and variables of tester class objects.	97
Figure B.1	The 2-dimensional representation of the loss function values when the anchor is located in (0.0, 0.0) and the positive is at (1.0, 0.0). The plot (A) shows the circle due to the first object, which is selected as the anchor of the triplet. Similarly, plot (B) represents the region when the object at (1.0, 0.0) is selected as the anchor point. The plot (C), on the other hand, shows the loss function values for the our loss function when $r = 0, s = 0, d = 1, \rho = 12/7, m_1 = 4.6381$. Finally, the plot (D) shows the overlapping area of the previous three plots.	101

LIST OF ABBREVIATIONS

R@K	Recall at K
P@K	Precision at K
P@R	R-Precision
MAP@R	Mean Average Precision at R
METU Loss	Metric Learning with Entangled Triplet Unified Loss

CHAPTER 1

INTRODUCTION

Object verification from visual data and image retrieval are two of the most popular computer vision problems. These problems are typically encountered while searching a similar image in a visual database. The aim of the algorithm in these fields is to measure the semantical similarity of any two objects in the images. Moreover, the methods proposed to solve these problems also enhance the performance of recognizing some unseen types of objects. Indeed, the image retrieval problem is regarded as a sub-problem of the classification. Although the definitions of these two problems vary from one to another, they are intrinsically related to each other.

In general, the classification task can be defined as follows: For a given image of an object whose class is known, a classifier tries to predict the true label of the object. In other words, the problem of classification focuses on the *class probabilities* of the samples and the class accuracy of the model. On the other hand, the aim of the image retrieval task is to learn the *features* of the objects in the images and distinguish the samples with these features.

Nevertheless, the methods proposed for solving image retrieval problems can be utilized as the feature extractors of the systems which are designed for classification task in order to improve the effectiveness of the classification algorithms. Obviously, if the first stage of the algorithm, which is a feature extractor, works fine, then the class accuracy of classification (the ratio between the correct predictions and the total number of predictions) will also increase. Therefore, the research efforts on these two topics influence the performance of each other.

1.1 Emerging of Deep Networks

There are plenty of studies focused on visual object classification via several techniques in machine learning. These techniques are applied to some challenging datasets in order to compare with each other. The ImageNet [5] is one of the most well-known challenges among them. The classical techniques such as Support Vector Machine (SVM) reached the highest performance score levels until the 2010s. However, a significant improvement was encountered in 2012 with AlexNet, which is inspired by LeNet (whose architecture consists of 4 layers as shown in Figure 1.1). Note that LeNet has only two layers that are convolutional, while the others are fully connected.

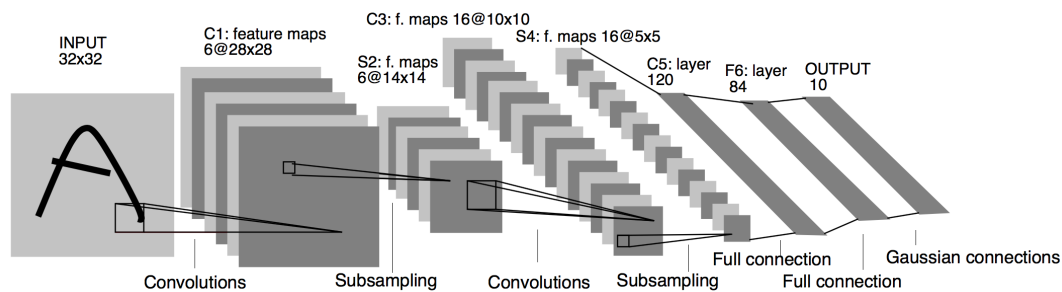


Figure 1.1: The architecture of LeNet. The figure is taken from the original paper [21].

On the other hand, AlexNet in Figure 1.2 has 8 layers. Thus, this approach based on neural networks is called *deep* learning. The popularity of AlexNet is from being the first deep learning based method that won the ImageNet image classification challenge.

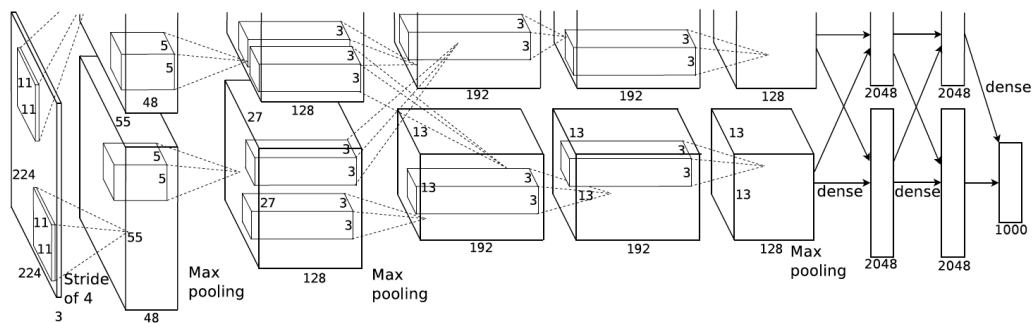


Figure 1.2: The architecture of AlexNet. (The figure from the paper [20])

After AlexNet, all the winners of the ImageNet challenge have utilized the deep learning based models [43, 35, 13]. The deep learning models dominate not only this challenge but also other challenges for various computer vision tasks. For example, the Visual Object Tracking (VOT) challenge [18] is about the object tracking task, and the deep learning based methods also lead this competition.

After ImageNet 2012 Challenge, some new network architectures were proposed. The main aim of designing these architectures is to solve the over-fitting problem that causes decreasing the performance score of the trained models. In order to avoid over-fitting, the parameters in the models are reduced without decreasing the discriminability capacity. There are mainly three milestone papers after AlexNet that gradually solve this issue: VGGNet [30], GoogleNet [35], and ResNet [13].

The first study VGGNet [30] introduces the receptive field concept, which is the size of the area in the image that a neuron in a filter can *see* (take into consideration while calculating the output value). Then, by using 3×3 filters, the number of parameters in the model can be reduced while increasing the depth. The second study, GoogleNet, proposes the inception module and increases the depth to 22 layers [35]. On the other hand, the last effort, ResNet in [13], shows the enhancement of the performance by utilizing the residual connections (named skip-connections as well). Although these prominent works have great successes, this field is still an active research area.

Likewise, for the image retrieval task, the deep metric learning based algorithms are also utilized due to their significant performances. These methods always utilize deep networks, i.e., similar to the network architectures in the classification task, but the loss functions are quite different. In this study, we focus on image retrieval on *fine-grained* datasets with deep metric learning techniques.

1.2 Deep Metric Learning

The aim of metric learning is to find out an embedding space and a distance function that any semantically similar instances are mapped in the nearby locations, and the remaining ones are located at further positions with respect to each other [15]. In order to solve this problem, several loss functions are proposed, such as contrastive

[2], and triplet losses [14]. However, one of the recent surveys about this area [26] argues that the accuracy scores taken by the studies in this field do not indicate the actual performance increment due to the flaws in their experimental methodologies. Hence, a framework should be designed in the first place to test all the methods that utilized the triplets to be fair against all these techniques.

As an initial observation, we notice a gap in the literature that the gradients of the loss function do not give the *logical* magnitude and direction. At this point, we try to exploit a well-known physical phenomenon: Electrostatic forces. The point charges try to minimize their potential energies with moving due to the electrostatic forces affecting them, which can be simply calculated by the partial derivatives of the potential energy of the system with respect to the coordinates of each particle. Likewise, we establish the analogies between these charged particles & the objects in the embedding space, the loss function value & the overall potential energy of the system, and the forces due to charges & the gradients of the loss function with respect to each anchor, positive, and negative items. Then, we determine the gradients that are affected on the items in a triplet and design a novel loss function inspired by this phenomenon.

1.3 Datasets for Image Retrieval

There are several new benchmark datasets that have been utilized in this field until 2012 in order to evaluate the performance of the models and the optimization techniques. Mainly, these datasets can be divided into two categories: Coarse-grained and fine-grained datasets. The coarse-grained ones include the samples that are in more than a single meta-class, in contrast to the fine-grained datasets. Hence, the performance of the models becomes lower on the fine-grained ones.

In this thesis, we conduct simulations on a common fine-grained dataset: CUB200-2011 [40], in addition to a synthetic dataset generated by Gaussian distribution, for showing the effectiveness of our novel method in comparison with the recent methods that also attack the retrieval problem by using the triplets. The performance scores of the models are calculated by the zero-shot learning scenario. Finally, the models trained with the proposed methods give promising results.

1.4 Problem Statement

In a mathematical perspective of the problem, let the given dataset D contains N training and $M - N$ test images, that is $D = D_{train} \cup D_{test} = \{X_i\}_{i=1}^N \cup \{X_i\}_{i=N+1}^M$, where $X_i = (x_i, y_i)$ denotes the i^{th} sample, $x_i \in \mathbb{R}^n$ is the vector of the pixel values, and $y_i \in \{1, \dots, L\}$ is the label. For the open-set split, the constraints, i.e. $y_i \neq y_j, \forall i \in \{1, \dots, N\}$ and $\forall j \in \{N + 1, \dots, M\}$ are satisfied. The network model is defined as the function $g_{\theta, \Lambda}(x_i) = f_i : \mathbb{R}^{(m+l)} \times \mathbb{R}^n \rightarrow \mathbb{R}^k$ where f_i is the embedding space coordinates of the i^{th} image, the $\theta \in \mathbb{R}^m$ denotes the trainable variables and the $\Lambda \in \mathbb{R}^l$ is for the hyper-parameters. This notation indicates that the model maps the image from n -dimensional space into k -dimensional embedding (or feature) space. The feature space can be defined as a metric space; then, there should be a distance defined on the k -dimensional space that satisfies the properties of the norms, denoted as $d(\cdot, \cdot) : \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}$.

The fundamental objective of metric learning can be defined as follows: For $\forall i, j, k \in N + 1, \dots, M$ and the sets $A_i = \{x_j | y_i = y_j\}$ and $B_i = \{x_k | y_i \neq y_k\}$, the inequality $d(g(x_i), g(x_j)) < d(g(x_i), g(x_k))$ where $x_j \in A_i$ and $x_k \in B_i$ should be satisfied.

Let the score function be able to measure the effectiveness of the model on test set (in order to display the generalizations performance) which is defined as $S_d(\cdot) : \mathbb{R}^{n \times (M-N)} \rightarrow \mathbb{R}$. The subscript d indicates the score function utilizes the distance function d during the calculation of the score value. Note that the higher score value can be evaluated from the better model. Then, the optimization problem can be described in Equation (1.1).

$$\max_{\Lambda} \left(\max_{\theta} S \left(\{g_{\theta, \Lambda}(x_i)\}_{i=N+1}^M \right) \right) \quad (1.1)$$

One of the trivial solutions for solving this optimization problem is to find the optimal $\hat{\theta}$ values from the second derivative of the score function. There are three reasons why one cannot use this solution:

- The test samples cannot be utilized in the training phase. If they are used, then the score value does not indicate the generalization performance of the model.

It is defined as cheating in the literature of deep learning.

- The score function may not be convex; hence, the point that the second derivative of the score function equals 0 and is a saddle or a local extremum point.
- The score function needs $M - N$ images to evaluate the score value, but this evaluation needs excessive memory that causes increasing the complexity of the model. Because of practical reasons, this technique cannot be exploited on large datasets.

In order to solve the first issue, the training images may be used for evaluating the score value as well. To elaborate, the training set can be split into two parts (train and validation sets) similar to the zero-shot scenario, and all the outermost max can be determined by the score calculated on the validation set, which is a standard method in deep learning. Also, this can be done by the cross-validation technique.

The second issue can be overcome by using iterative optimization methods such as steepest descent. Generally, the first-order derivative based iterative methods are used with the concept of backpropagation for training the deep neural networks due to their low computational requirements. The stochastic gradient descent (SGD), RMSprop, and Adam optimizers are also frequently utilized techniques in this field. This optimizers can determine the $\hat{\theta}$ but not Λ . In order to determine the hyper-parameters, the various search algorithms may be exploited, such as grid search, random search, and Bayesian optimization.

The last issue is about the computational complexity of the evaluation. All of the datasets cannot be taken into account in one step of optimization. Hence, the mini-batch concept may be utilized for coping with this problem. This solution also creates its own problems. For instance, when the batch size is small, the optimizer cannot find the local optimum location for the parameter set since it cannot take into account the sufficient number of samples in the dataset and calculates the incorrect updates in each batch. However, when the batch size is selected large enough, each batch can generalize the overall distribution in the dataset.

After using the aforementioned solutions, the overall optimization problem reduces to a sub-problem: What is the optimal (or sub-optimal) update for each sample in

a mini-batch so that the embedding space is transformed into a new form that gives better scores?

This optimal update can be adjusted directly with the help of designing the loss function in the framework. Another (indirect) way to modify the update is to use a miner for selecting only the *important* samples in the train set during the training phase. The instruments one can utilize while designing the framework are as follows: The optimizers in deep learning are calculating the first derivative of the loss functions in order to evaluate the backpropagation weights. In addition, any embedding vector of a sample in the train set can be found while evaluating the loss value. The answer to the question (if it exists) should utilize only these two pieces of information to form the update.

1.5 Motivation

In this work, the studies in deep metric learning, especially on triplet-based methods, are rigorously investigated with the framework implemented by us. To the best of our knowledge, there is a gap in the literature that no logical explanation about the relationship between the loss function design and the performance scores is stated. In the light of the observations about the simulations performed on our framework, we notice that one can make an analogy between the deep metric learning loss and the electrostatic forces. By extending this analogy, we analyze the losses in the literature and regard their flaws. In addition, we propose the losses that cope with these shortcomings in order to enhance the effectiveness of the model.

1.6 Contributions

Our contributions are three-fold:

- A novel loss function is proposed and its gradients are analyzed for the different cases. The motivation behind the loss is also explained.
- We introduce a framework for examining the loss functions in a fair manner.

- The proposed loss function is utilized in several experiments in our framework and compared with some recent methods from the relevant literature through experiments. We show the proposed method surpasses the performance of the methods in the same category.

CHAPTER 2

AN OVERVIEW ON DEEP METRIC LEARNING

In this chapter, we mainly analyze the deep metric learning techniques in the literature. The methods are explained in chronological order in Section 2.1 so that the progress of this field can be realized clearly. Moreover, the performance scores of these methods are just as important as the mechanisms of them since they point out whether the sub-problem solved by the method causes significant improvement or not. Therefore, we mention the performance scores in Section 2.2, what they measure, and the evaluation formulas. Moreover, one of the most important factors during a fair evaluation of the algorithms of deep metric learning is to select the dataset on which they are trained. This constraint is due to the fact that the performance scores cannot be meaningful when the discriminability of the dataset increases. On the other hand, when the model is trained on an extremely difficult dataset to discriminate, the scores might again be pointless. Thus, we investigate the benchmark datasets utilized for sorting these algorithms in Section 2.3.

2.1 State-of-the-Art Methods of Deep Metric Learning

The aim of deep metric learning is to separate the semantically irrelevant patterns apart from each other and map similar images into the near locations in the embedding space. For image data, it is utilized in many different tasks, such as image classification and image retrieval problems.

Such a sample-by-sample comparison process is intuitive; however, there are two main disadvantages, as well as four major advantages of these methods. The disad-

vantages can be stated as follows:

- The training process requires more resources than the standard loss functions, such as cross-entropy loss. Any training items are used more than once in each epoch in order to compare the coordinates in the embedding space. For example, although the cross-entropy loss function calculates the loss value from only one sample and its ground truth label, the triplet-based losses use embeddings of the three samples for the construction of a triplet. Hence, the batch size of the metric learning losses is smaller than the others due to this reason.
- The training duration might increase due to the smaller batch size. The memory allocation limits the size of each batch and the time of each epoch. Moreover, the metric learning algorithms might need a mining stage before each epoch in order to select the most valuable samples from the dataset. Then, it also leads to a loss of time.

Apart from these disadvantages, the methods have been still utilized due to their advantages itemized below:

- The models trained with the metric learning methods can learn the feature space rather than learning only the class in the datasets. Hence, they might be more robust for variation of the classes and the changes of the style of images, since metric learning losses avoid the models to overfit the data, but the features, inherently. These losses force the model to learn the features of the data by comparing the other data in the dataset.
- Unlike the generic loss functions that utilize only the predicted probability distribution on a single sample once, the metric learning loss functions utilize the information of the relationship between the samples. For instance, the triplet loss function takes account of both the anchor, positive, and negative samples' features in each update of the gradients.
- The metric learning losses are more robust for overfitting the data; they can deal with the outliers with the help of mining methods. Moreover, it is possible to remove the effect of an outlier by comparing it with the other inlier samples.

- Metric learning methods can be exploited for the *zero-shot learning scenario*, i.e., learning without any representative sample for a particular class. This scenario is quite frequently encountered in real-world applications. Moreover, these methods have the power of generalization rather than the standard methods due to the fact that they utilized the information of the relationship between the samples in addition to themselves and their ground truth labels.

The state-of-the-art methods for deep metric learning are explained in the subsequent subsections. The critical paradigms for deep metric learning can be divided into the following cases:

- Loss functions,
- Mining techniques,
- Advanced methods.

2.1.1 Loss Functions in Metric Learning

There are different types of loss functions in metric learning that are utilized for various purposes. In the classification and retrieval tasks, the most popular losses can be examined in the following categories:

- Classification loss,
- Triplet-based loss,
- Batch-based loss.

The first type of loss is the standardized loss function for the classification tasks. However, classification loss cannot reach the performance scores of the others due to the fact that these types of losses only utilize the class information of a single image for a single update, and they miss the information between the samples. The second type of loss uses only the triplets rather than all the samples in the batch. Nevertheless, the triplet-based losses should be analyzed in order to understand the

batch-based losses because the basic properties of the losses can be realized easier in the tuple-based losses compared to the batch-based ones. Moreover, any tuple-based loss can be generalized for constructing a batch-based loss. In other words, the tuple-based losses can be thought of as the core of the batch-based ones.

2.1.1.1 Classification Losses

Classification is one of the most popular tasks in machine learning, as mentioned in Chapter 1. From the deep learning perspective, there are several network architectures that are proposed for solving this popular problem with higher accuracy. The well-known *cross-entropy loss* with softmax probability function is the most frequently utilized loss function in this task [8]. Frequent utilization of cross-entropy loss is due to two reasons: Cross-entropy is a quite efficient way to compute losses since the loss function needs only the probability mass vector of a given image (named as *logits*) and its ground truth label. The second reason is due to the fact that this loss can reach the highest scores without any other auxiliary information from the samples.

It should be noted that the cross-entropy loss function simply evaluates the *Kullback–Leibler divergence* (KL-divergence) between predicted and the true labels. If the distance between them is large, the loss function gives more penalty to the model for updating the parameters. The relation for the cross-entropy loss is given in (2.1),

$$\mathcal{L}_{x-entropy}(\xi, y) = -\log \frac{e^{\xi_y}}{\sum_{i=1}^N e^{\xi_i}}, \quad (2.1)$$

where N is the number of unique labels, y is the true label, and the ξ_i is the value of i^{th} element of the logit ξ , which is equal to the predicted probability for the image that is belonging to the i^{th} class.

An extension to this loss function can be obtained by using a *temperature* parameter while calculating the loss value [1]. This loss function is presented below:

$$\mathcal{L}_{temperature}(\xi, y) = -\log \frac{e^{\xi_y/T}}{\sum_{i=1}^N e^{\xi_i/T}}, \quad (2.2)$$

where T is the temperature value. It should be noted that Equation (2.1) is the special case of $T = 1$ of Equation (2.2). The effect of the temperature parameter is to adjust the confidence of the model for determining the labels of the samples. When the T

value is high, the predicted distribution becomes flatter, the confidence reduces. On the other hand, the smaller T value leads to the sharper predicted distribution. This flexibility might increase the performance of the models in some of the cases.

2.1.1.2 Tuplelet-based Losses

Prior research on deep metric learning suggests that there should be a feature extractor model that can be trained by pairs selected from the training set. The feature extractor is expected to learn the features predefined on the dataset on which the model performed, such as the nose and eye regions in a face image dataset. However, defining these features takes time and is not efficient for huge datasets due to the labeling of the enormous number of samples as mentioned in Section 2.3.2. Then, the main idea of deep metric learning becomes training the model for learning the *abstract features* by the help of comparing the samples in the dataset.

Taking these ideas into account, the Siamese nets [2] were proposed in the 1990s with a novel *contrastive loss* shown below:

$$\begin{aligned} \mathcal{L}_{contrast} = & \delta[y_i - y_j] \cdot \|g_\theta(x_i) - g_\theta(x_j)\|_2^2 \\ & + (1 - \delta[y_i - y_j]) \cdot \max(0, \alpha - \|g_\theta(x_i) - g_\theta(x_j)\|_2^2). \end{aligned} \quad (2.3)$$

The term α and $\max(0, \cdot)$ function are utilized for preventing the model from fluctuation. That is, if there was no such hyper-parameter, all the negative pairs generate the gradients even if they are at distant locations to each other. With the help of these terms, the negative pairs, whose labels are not the same, are not penalized if the distance between them is greater than α . On the other hand, the model may still suffer from the collapsing issue since all the samples in the same class force to be mapped into a single location in the embedding space. It causes the model to ignore the information of intra-class features; hence, the samples from the unseen classes would not be distinguished. The model would not be robust for over-fitting into the train set, which will be discussed after introducing the triplet loss.

As the name suggests, there are two (twin) networks sharing their parameters with each other as shown in Figure 2.1. The difference between the embedding values of pairs is calculated from the last layer of the models.

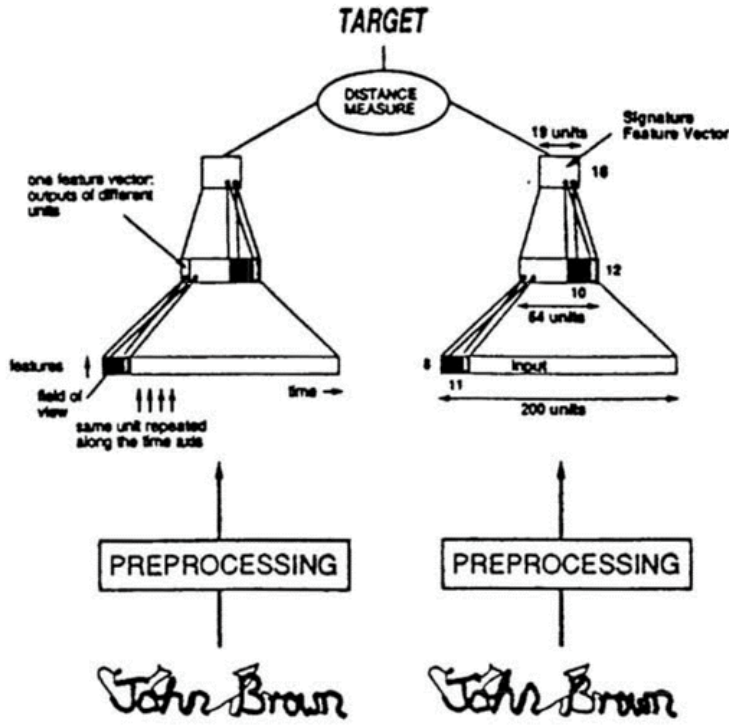


Figure 2.1: The Siamese networks [2] are proposed for solving the problem of determining whether the signatures in the given two images are the same or not. The trainable parameters of the models in this figure are shared (i.e., they are equal) with each other. They extract the features of the images in different time intervals; next, embedding values for these images are compared to each other. The figure is taken from the paper [2].

There are subsequent studies [14] that are inspired from the Siamese nets [2] extending this framework by utilizing *triplets* shown below:

$$\mathcal{L}_{triplet} = \left(\frac{e^{\|f_a - f_p\|_2}}{e^{\|f_a - f_p\|_2} + e^{\|f_a - f_n\|_2}} \right)^2, \quad (2.4)$$

where f_a , f_p , and f_n are represents the anchor, positive and negative items, typically vectors. In a valid triplet, the inequality $y_a = y_p \neq y_n$ must hold.

The overall framework of the triplet-based models is shown in Figure 2.2. At this point, it is important to note that there are only two pairwise distances, $f_a \leftrightarrow f_p$ and $f_a \leftrightarrow f_n$, utilized in this and subsequent losses by a triplet.

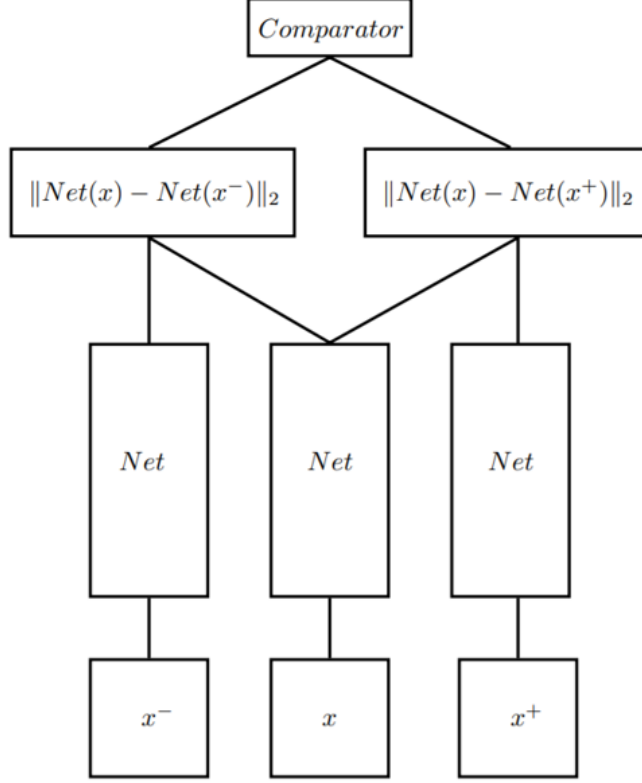


Figure 2.2: Similar to the Siamese networks [2] shown in Figure 2.1, all the three networks share their trainable parameters to each other (figure of the triplet network is taken from [14]).

By using the same framework in Figure 2.2, a *ranking loss* is proposed in the study [36]. In addition, the study in [42] has already suggested utilizing the same loss function 3 years before the work in [36].

$$\begin{aligned} \mathcal{L}_{ranking}^{hinged} &= \max(0, \mathcal{L}_{ranking}), \\ \mathcal{L}_{ranking} &= \|f_a - f_p\|_2 - \|f_a - f_n\|_2 + m. \end{aligned} \tag{2.5}$$

The aim of the proposed ranking loss is the same as the triplet loss in (2.4), which is the Euclidean distance between f_a and f_p should be smaller than the one between f_a and f_n .

Although the original triplet loss in (2.4) inspired from the cross-entropy loss with softmax function, the ranking loss and the extensions of it utilize the *hinge function* (or *ramp function*, denoted as $\max(0, \cdot)$). Preference of this function is due to avoid-

ing the model to collapse, i.e., if there is no such function, all the samples with the same label will be mapped into the same location in the embedding space.

The hinge function and its hyper-parameter denoted as m inhibit the loss, if $\|f_a - f_p\|_2 + m < \|f_a - f_n\|_2$; then, the samples with the same label are mapped into a hyper-sphere whose radius is at least m (since $\|\cdot\| : \mathbb{R}^m \rightarrow \mathbb{R}^{\geq 0}$). Moreover, the hinge function has brought further benefits rather than avoiding it. A study [42] points out the difference between pair-based losses and the triplet losses as follows: Although there should be a predefined threshold, m in (2.3), in pair-based loss functions, the triplet loss functions have more flexibility to adapt their thresholds, denoted as $m + \|f_a - f_p\|_2$ in (2.5), by using the intra-class variance of the different classes (the second term in the latter expression).

At this point, the gradients of these loss functions should also be examined. The partial derivative of the distilled form of the ranking loss function ($\mathcal{L}_{ranking}$ in Equation (2.5)) can be obtained as:

$$\begin{aligned}\frac{\partial \mathcal{L}_{ranking}}{\partial f_a} &= \frac{f_a - f_p}{\|f_a - f_p\|_2} - \frac{f_a - f_n}{\|f_a - f_n\|_2}, \\ \frac{\partial \mathcal{L}_{ranking}}{\partial f_p} &= \frac{f_p - f_a}{\|f_a - f_p\|_2}, \\ \frac{\partial \mathcal{L}_{ranking}}{\partial f_n} &= \frac{f_a - f_n}{\|f_a - f_n\|_2}.\end{aligned}\tag{2.6}$$

By the chain rule, the derivatives of original loss can be simply obtained by the help of $\frac{\partial \mathcal{L}_{ranking}^{hinged}}{\partial f_i} = \frac{\partial \mathcal{L}_{ranking}^{hinged}}{\partial \mathcal{L}_{ranking}} \cdot \frac{\partial \mathcal{L}_{ranking}}{\partial f_i}$ where $\frac{\partial \mathcal{L}_{ranking}^{hinged}}{\partial \mathcal{L}_{ranking}}$ is equal to Heaviside step function (or unit step function). That is it gives 1 if $\mathcal{L}_{ranking} > 0$; otherwise 0.

Another proposed loss by using triplets, namely *FaceNet* [28], has a slight modification, whose relation is shown below (please note the norm square, ℓ_2^2):

$$\begin{aligned}\mathcal{L}_{facenet}^{hinged} &= \max(0, \mathcal{L}_{facenet}), \\ \mathcal{L}_{facenet} &= \|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + m.\end{aligned}\tag{2.7}$$

The equation in (2.7) also equals to $\mathcal{L}_{facenet} = 2 \cdot (f_a^T \cdot f_n - f_a^T \cdot f_p) + \|f_p\|_2^2 - \|f_n\|_2^2 + m$. Note that if the embedding vectors, f_i 's, are ℓ_2 normalized, i.e. $\|f_i\|_2 = 1, \forall i$, then the expression becomes $\mathcal{L}_{facenet} = 2 \cdot (f_a^T \cdot f_n - f_a^T \cdot f_p) + m$. At that point, the term of $f_i^T \cdot f_j$ resembles the inner product (non-normalized cosine similarity function).

The authors [28] claim that although there is a small difference between the cost relations, the results are significantly improved when this loss function is used with *semi-hard miner*, to be explained in Section 2.1.2. Another study [42] argues that the FaceNet loss function might suffer from the collapsing issue when the hard-negative miner is used; thus, the performance of the ranking loss should surpass that of FaceNet loss. All in all, the results of *FaceNet* [28] show the effectiveness of its loss function, which cannot be outperformed by other methods.

Although it is similar to the ranking loss in (2.6), the partial derivatives of the FaceNet loss function below is quite different from (2.6):

$$\begin{aligned}\frac{\partial \mathcal{L}_{facenet}}{\partial f_a} &= f_n - f_p, \\ \frac{\partial \mathcal{L}_{facenet}}{\partial f_p} &= f_p - f_a, \\ \frac{\partial \mathcal{L}_{facenet}}{\partial f_n} &= f_a - f_n.\end{aligned}\tag{2.8}$$

Later, the ratio loss [41] is proposed as well, whose distilled form is given below:

$$\mathcal{L}_{ratio} = 1 - \frac{\|f_a - f_n\|_2}{\|f_a - f_p\|_2 + m}.\tag{2.9}$$

The hinge function can also be exploited while calculating the overall loss value, since the value of the distilled form may be negative when $\|f_a - f_p\| + m < \|f_a - f_n\|$, similar to (2.5). The equation in (2.9) uses the distance between $f_a \leftrightarrow f_p$ in the denominator, similar to the original triplet loss in (2.4). Taking the derivatives of the loss function gives the following equality relations:

$$\begin{aligned}\frac{\partial \mathcal{L}_{ratio}}{\partial f_a} &= \frac{1}{\|f_a - f_p\|_2 + m} \cdot \left(\frac{\|f_a - f_n\|_2 \cdot (f_a - f_p)}{\|f_a - f_p\|_2} - \frac{\|f_a - f_p\|_2 \cdot (f_a - f_n)}{\|f_a - f_n\|_2} \right), \\ \frac{\partial \mathcal{L}_{ratio}}{\partial f_p} &= \frac{1}{\|f_a - f_p\|_2 + m} \cdot \left(\frac{\|f_p - f_n\|_2 \cdot (f_p - f_a)}{\|f_a - f_p\|_2} \right), \\ \frac{\partial \mathcal{L}_{ratio}}{\partial f_n} &= \frac{1}{\|f_a - f_p\|_2 + m} \cdot \left(\frac{f_a - f_n}{\|f_a - f_n\|_2} \right).\end{aligned}\tag{2.10}$$

It should be noted that the original triplet loss' derivatives in (2.11) is significantly

different from Equation (2.10).

$$\begin{aligned}
\frac{\partial \mathcal{L}_{triplet}}{\partial f_a} &= \frac{2 \cdot e^{2\|f_a - f_p\|_2 + \|f_a - f_n\|_2}}{(e^{\|f_a - f_p\|_2} + e^{\|f_a - f_n\|_2})^2} \cdot \left(\frac{f_a - f_p}{\|f_a - f_p\|_2} - \frac{f_a - f_n}{\|f_a - f_n\|_2} \right), \\
\frac{\partial \mathcal{L}_{triplet}}{\partial f_p} &= \frac{2 \cdot e^{2\|f_a - f_p\|_2 + \|f_a - f_n\|_2}}{(e^{\|f_a - f_p\|_2} + e^{\|f_a - f_n\|_2})^2} \cdot \frac{f_p - f_a}{\|f_a - f_p\|_2}, \\
\frac{\partial \mathcal{L}_{triplet}}{\partial f_n} &= \frac{2 \cdot e^{2\|f_a - f_p\|_2 + \|f_a - f_n\|_2}}{(e^{\|f_a - f_p\|_2} + e^{\|f_a - f_n\|_2})^2} \cdot \frac{f_a - f_n}{\|f_a - f_n\|_2}.
\end{aligned} \tag{2.11}$$

In fact, the directions of the gradients are quite similar to the ones of ranking loss in (2.6), but it is important that the original triplet loss does not use the hinge function like the ranking loss.

Noting the similarities between losses for unit magnitudes, some metric losses with cosine similarity have also been developed. The initial study, namely *angular loss* [37], tries to utilize the angular similarity between the tuplets, as given below:

$$\mathcal{L}_{angular} = \|f_a - f_p\|_2^2 - 4 \tan^2(\rho) \cdot \left\| f_n - \frac{f_a + f_p}{2} \right\|_2^2. \tag{2.12}$$

This loss function also uses the hinge function to avoid collapsing the model. The partial derivatives are shown in (2.13).

$$\begin{aligned}
\frac{\partial \mathcal{L}_{angular}}{\partial f_a} &= 2 \cdot (f_a - f_p) - 2 \tan^2(\rho) \cdot (f_a + f_p - 2f_n), \\
\frac{\partial \mathcal{L}_{angular}}{\partial f_p} &= 2 \cdot (f_p - f_a) - 4 \tan^2(\rho) \cdot (f_a + f_p - 2f_n), \\
\frac{\partial \mathcal{L}_{angular}}{\partial f_n} &= 4 \tan^2(\rho) \cdot (f_a + f_p - 2f_n).
\end{aligned} \tag{2.13}$$

Some recent studies focus on adding a regularization constant to the metric loss in order to get high performance from training. Later, *moving loss* [3] is proposed, whose loss function without the hinge function is shown in Equation (2.14).

$$\mathcal{L}_{moving} = \|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 - \rho \cdot \frac{1 - f_p^T f_a}{\|f_a - f_p\|_2 - \|f_a - f_n\|_2} + m. \tag{2.14}$$

In addition, the gradients of the moving loss in (2.14) are given below for completeness:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{moving}}{\partial f_a} &= 2 \cdot (f_n - f_p) - \rho \cdot \left(\frac{f_a - f_p}{\|f_a - f_p\|_2 \cdot \|f_a - f_n\|_2} + \frac{\|f_a - f_p\|_2}{\|f_a - f_n\|_2^3} \cdot (f_n - f_a) \right), \\
\frac{\partial \mathcal{L}_{moving}}{\partial f_p} &= 2 \cdot (f_p - f_a) - \rho \cdot \frac{f_p + f_a}{\|f_a - f_p\|_2 \cdot \|f_a - f_n\|_2}, \\
\frac{\partial \mathcal{L}_{moving}}{\partial f_n} &= 2 \cdot (f_a - f_n) - \rho \cdot \frac{f_a + f_n}{\|f_a - f_p\|_2 \cdot \|f_a - f_n\|_2^3}.
\end{aligned}
\tag{2.15}$$

2.1.1.3 Batch-based Losses

Batch-based losses are designed for utilizing all the samples in a batch while calculating the loss value. Therefore, they use more information and get the *smarter* updates without an additional computation time. This situation is described in [33] as follows: "adding extra vertices to the graph is a lot more costly than adding extra edges because adding vertices to the graph incurs extra I/O time and/or storage overhead." This explanation indicates after calculating the embeddings of the images in a single batch, there is almost no additional computational power needed for the operation of the embeddings while calculating the loss function.

Although this statement is true for almost all the popular platforms, such as PyTorch [27], the triplet losses do not have to suffer from this limitation as well. Indeed, the study in [26] suggests that using a framework with all the valid triplets can be obtained in a single batch that is exploited for training, whose number is roughly 10^6 , if the batch size is 32. Nevertheless, the progress in the literature for this type of loss function will be explained next.

The pioneering work in batch-based loss approaches is *lifted structure loss* [33]. The selection of the tuples in each batch is represented in Figure 2.3.

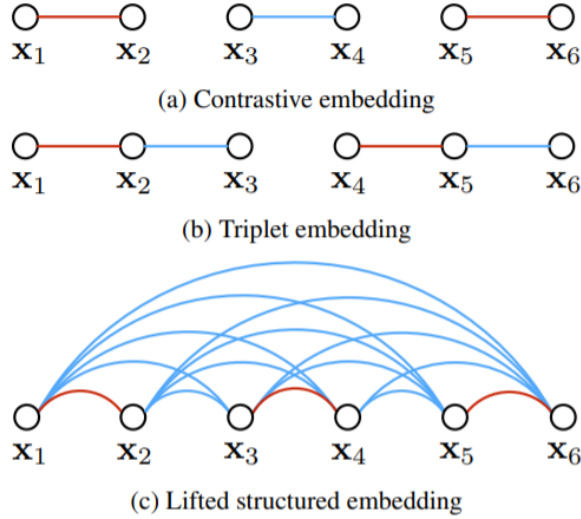


Figure 2.3: Selection the tuples for lifted structure loss (Figure from [33]).

The lifted structure loss function is presented below:

$$\mathcal{L}_{lifted}^{hinged} = \frac{1}{2|P|} \sum_{(i,j) \in P} \max(0, \tilde{J}_{i,j})^2,$$

$$\tilde{J}_{i,j} = \log \left(\sum_{(i,k) \in N} e^{m-D_{i,k}} + \sum_{(j,l) \in N} e^{m-D_{j,l}} \right) + D_{i,j}, \quad (2.16)$$

$$D_{1,2} = \|f_1 - f_2\|_2,$$

where $P = \{(i, j) | y_i = y_j\}$, $N = \{(i, j) | y_i \neq y_j\}$, and $|\cdot|$ is the cardinality function, which gives the number of element in a set.

These types of losses are not used tuple of anchor, positive and negatives, but one can analyze them as if they take a triplet in a batch in order to compare them with the triplet-based losses. Therefore, $\tilde{J}_{i,j}$ cost equals to $\tilde{J}_{a,p}$ (or \mathcal{L}_{lifted} similar to the triplet-based losses), and $N = \{(a, n), (p, n)\}$. The triplet version of lifted structure loss is given in Equation (2.17).

$$\mathcal{L}_{lifted}^{hinged} = \max(0, \log(e^{m-D_{a,n}} + e^{m-D_{p,n}}) + D_{a,p}). \quad (2.17)$$

Then, the gradients of the lifted structure loss function are obtained as (2.18).

$$\begin{aligned}
\frac{\partial \mathcal{L}_{lifted}}{\partial f_a} &= \frac{f_n - f_a}{\|f_a - f_n\|_2} \cdot \frac{e^{m - \|f_a - f_n\|_2}}{e^{m - \|f_a - f_n\|_2} + e^{m - \|f_p - f_n\|_2}} + \frac{f_a - f_p}{\|f_a - f_p\|_2}, \\
\frac{\partial \mathcal{L}_{lifted}}{\partial f_p} &= \frac{f_n - f_p}{\|f_p - f_n\|_2} \cdot \frac{e^{m - \|f_p - f_n\|_2}}{e^{m - \|f_a - f_n\|_2} + e^{m - \|f_p - f_n\|_2}} + \frac{f_p - f_a}{\|f_a - f_p\|_2}, \\
\frac{\partial \mathcal{L}_{lifted}}{\partial f_n} &= \frac{\frac{f_a - f_n}{\|f_a - f_n\|_2} \cdot (e^{m - \|f_a - f_n\|_2}) + \frac{f_p - f_n}{\|f_p - f_n\|_2} \cdot (e^{m - \|f_p - f_n\|_2})}{e^{m - \|f_a - f_n\|_2} + e^{m - \|f_p - f_n\|_2}}.
\end{aligned} \tag{2.18}$$

Another loss inspired from [33] is *N-pairs loss* [31]. The sampling strategy of this study is shown in Figure 2.4.

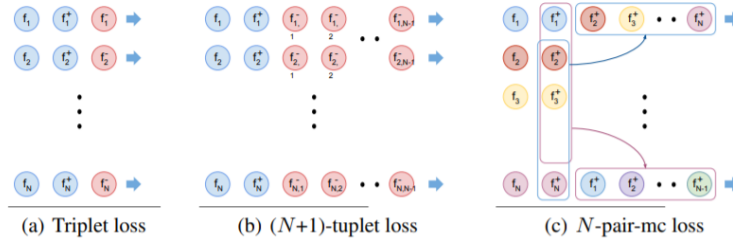


Figure 2.4: A mini-batch for N-Pairs loss function. (a), (b), and (c) show a batch for triplet, (N+1)-tuple and N-pairs losses, respectively (Figure from [31]).

Two different versions of N-pairs loss is also proposed in the same paper [31]. The first one is *multi-class version of the loss function* (MC):

$$\begin{aligned}
\mathcal{L}_{npairs}^{mc} &= \frac{1}{B} \sum_{i=1}^B \log \left(1 + \sum_{j \neq i} e^{f_i^T f_j^+ - f_i^T f_j^-} \right), \\
\mathcal{L}_{npairs}^{mc} &= -\frac{1}{B} \sum_{i=1}^B \log \left(\frac{e^{f_i^T f_j^+}}{e^{f_i^T f_j^+} + \sum_{j \neq i} e^{f_i^T f_j^-}} \right),
\end{aligned} \tag{2.19}$$

where B is the batch size, f_i , f_i^+ and f_j^- represent the embeddings of anchor, positive and the negatives (more than one negative item are included in a single evaluation of the original form of N-pairs loss).

As the name implies, this loss function is quite similar to a multi-class logistic loss function, i.e., the cross-entropy loss with softmax probabilities, mentioned in Section 2.1.1.1. It is the main difference between the N-pairs and lifted structure loss, which is utilized the max-margin formulation [31].

The second version of N-pairs loss is slightly different from the first one. The proposed *one-vs-one* (OVO) version of the loss function is also shown below:

$$\mathcal{L}_{npairs}^{ovo} = \frac{1}{B} \sum_{i=1}^B \sum_{j \neq i} \log \left(1 + e^{f_i^T f_j^+ - f_i^T f_i^+} \right), \quad (2.20)$$

where B denotes the batch size.

Note that when $N = 2$, there is no difference between the equations in (2.19) and (2.20). Indeed, a triplet version of this loss function can be written as below:

$$\mathcal{L}_{npairs}^{triplet} = \log \left(1 + e^{f_a^T f_n - f_a^T f_p} \right). \quad (2.21)$$

The gradients of this cost are given below:

$$\begin{aligned} \frac{\partial \mathcal{L}_{npairs}^{triplet}}{\partial f_a} &= \frac{e^{f_a^T f_n - f_a^T f_p}}{1 + e^{f_a^T f_n - f_a^T f_p}} \cdot (f_n - f_p), \\ \frac{\partial \mathcal{L}_{npairs}^{triplet}}{\partial f_p} &= -\frac{e^{f_a^T f_n - f_a^T f_p}}{1 + e^{f_a^T f_n - f_a^T f_p}} \cdot f_p, \\ \frac{\partial \mathcal{L}_{npairs}^{triplet}}{\partial f_n} &= \frac{e^{f_a^T f_n - f_a^T f_p}}{1 + e^{f_a^T f_n - f_a^T f_p}} \cdot f_a. \end{aligned} \quad (2.22)$$

2.1.2 Mining Techniques

Many researchers argue that the sampling strategies are quite important and significantly enhance the performance scores of the trained models [42]. This result is due to the fact that there are an enormous amount of samples can be generated from a dataset. Moreover, these research efforts focus on selecting more significant samples (pairs, triplets, etc.) from such datasets.

In fact, if the number of the items in a dataset is N , the number of valid triplets might be the order of N^3 , which is infeasible for training [42]. For example, it is roughly $\sim 10^9$ triplets that can be generated from the CUB-200-2011 dataset, which has $\sim 10^4$ images. Besides, the majority of these samples do not enhance the training, which is denoted as *easy samples* in [6]. Therefore, some mining, i.e., data selection, methods should be proposed for solving this problem and improve the performance scores.

The study in [29] asserts that using the hard negative samples (i.e., the most similar samples to the anchor from other classes) improves the effectiveness of the training

since the randomly selected samples typically make the training harder and slower. Hence, this study proposes the *hard negative mining* which is shown below:

$$\hat{n}_{hard} = \arg \min_n \|f_a - f_n\|_2. \quad (2.23)$$

Note that if this searching process is performed on the whole training set, it is denoted as *offline mining* by [28].

Likewise, the hard positives can be defined as $\hat{p}_{hard} = \max_p \|f_a - f_p\|_2$, which is used in some studies, such as [29]. However, some researchers [42] argue that mining the positive and negative samples in the whole set of training is quite infeasible. On the other hand, a later study [28] shows selecting semi-hard samples makes the training more robust to the outliers. An *online mining* method is proposed in [28], i.e. the searching process is performed only on a mini-batch for each update. As mentioned in [42], *semi-hard negative mining* can be proposed as follows:

$$\begin{aligned} \hat{n}_{semihard} &= \arg \min_n \|f_a - f_n\|_2, \\ \text{such that } &\|f_a - f_n\|_2 > \|f_a - f_p\|_2. \end{aligned} \quad (2.24)$$

However, FaceNet algorithm [28] uses all negative samples which satisfy the inequality $\|f_a - f_n\|_2 > \|f_a - f_p\|_2$. Note that the same situation could have been created for [28] by simply modifying its loss function without utilizing the mining process in (2.24) as follows:

$$\mathcal{L}_{facenet}^{mined} = \min(\max(0, \mathcal{L}_{facenet}), m), \quad (2.25)$$

even if it is not stated in the paper [28]. The more detail about this modification is discussed in 3.2.3.

In this way, FaceNet loss in (2.7) becomes equal to (2.25) so that the loss intrinsically mines the semi-hard negative samples.

All these mining methods are illustrated in Figure 2.5. There is an important point to mention in this figure. When online mining is utilized, the method used for selecting negative samples restricts the mining of positive samples. For instance, there is no hard-negative-semi-easy-positive mining. If the negatives are selected in the hard region, all the positives are in the hard region as well.



(a) The regions for the negative samples.

(b) The regions for the positive samples.

Figure 2.5: 2-D representation of the embedding space. The margin, m , is a hyper-parameter of the miner. The coordinates of the samples A , P , and N represent the feature vectors of f_a , f_p , and f_n , which are embeddings of anchor, positive, and negative items of a triplet, respectively. Region #1 belongs to the *easy samples*, which do not create any gradient for margin-based losses, such as ranking loss. Region #2 is used for *semi-hard* mining. Finally, the items in Region #3 and #4 are accepted as *hard samples*.

Figure 2.5 shows there is still room for improvement for triplet mining methods. For example, the negatives and the positives selecting in the regions of #2 and #3 can be used for mining. It may prevent the new miner from being unable to find the samples in the target region, as encountered in the semi-hard miner mentioned in [28, 42].

Moreover, the distance weighted sampling [42] is proposed for avoiding the noises in the gradients of the updates caused by triplet loss. This study [42] shows when any two samples are close to each other, the gradients are more sensitive to noise, which is resulted from the small fluctuation in the pixel values. Hence, the model trained with hard negative mining becomes more fragile due to this effect. The weight, $w(\cdot)$, in Equation (2.26) are multiplied to the samples located in f_1 and f_2 in the embedding space.

$$w(\|f_1 - f_2\|_2) = \|f_1 - f_2\|_2^{2-m} \left(1 - \frac{\|f_1 - f_2\|_2^2}{4}\right)^{-1}. \quad (2.26)$$

Another study [11] suggests utilizing an effective method for finding the nearest neighbors in the embedding space and an adaptive controller for adjusting the hyper-

parameters of the proposed miner. This method mainly focuses on reducing the computational cost of the mining methods. Moreover, the effort explained in [46] tries to control the hardness level of the triplets with a generator. The label-preserving synthetic data is generated by this generator, which is trained in an adversarial manner. The adversarial training might create additional issues that are out of the scope of this thesis.

2.1.3 Advanced Methods

There is plenty of effort which are not included in the previous categories of the literature. All these works focus on select the meaningful negative and positive items in order to adjust the gradients acting on the anchor, which leads to having a higher performance score. One example of these works is called hierarchical triplet loss [7]. It tries to establish a link between intra-class variance and the margin like the m in (2.5). In this work, the classes are located in a tree structure in a hierarchical way, and the margin in the loss function for the similar classes is less than the one for the dissimilar ones. Another study [6] is to utilize the generator for generating some synthetic hard negatives from the easy ones. In addition, the proxy-based models [25, 16] also reach higher accuracy scores in recent years. They try to use the global distribution of the dataset while calculating each mini-batch update. The embedding vectors of the proxies provide this information about the distribution. To elaborate, they are updated in order to represent the so-called means of the classes in the train set. After that, the gradients of any anchor are evaluated with the help of these proxies.

Although the scores of these studies reach high values, their origins still depend on the tuple-based losses. It means, if the gradients of a triplet are adjusted optimally, the proxy or generative based methods also improve their performances.

2.2 Metrics for Performance Analysis

There could be many performance scores that are exploited in different problems. The scope of this thesis is limited to the scores of the following two problems: Image classification and retrieval. The metrics related to these two problems are presented

in Figure 2.6.

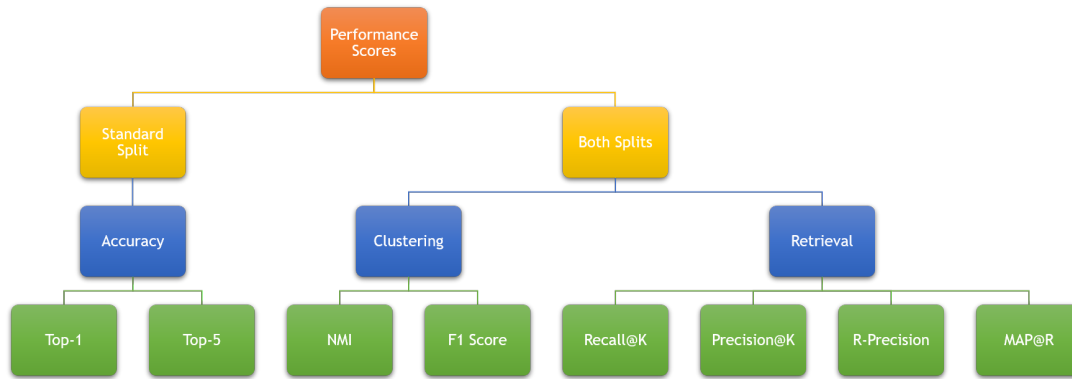


Figure 2.6: The tree for evaluation scores that are frequently used in fine-grained image classification and retrieval.

In Figure 2.6, the class accuracy scores can be utilized for the supervised learning problems. These scores need the class probabilities (called as *logits* in the literature) that are predicted by the trained model and the ground truth labels of each sample in the test set for calculating the values. It is generally used for evaluating classification tasks.

On the other hand, the clustering and retrieval scores are used for both unsupervised and supervised learning problems. The clustering scores measure the capability of the trained model to separate the samples belonging to the different classes in the dataset into the different coordinates in the embedding space. Generally, there is an unsupervised clustering step such as *K-Means* before evaluating the scores. Some studies [26] argue that this step may create unfair results due to the non-standardized initialization of the clustering algorithms and gives non-reliable values due to this reason. Then, these types of scores may give completely unreliable values for the same configuration with different initialization.

Another type of score is called the retrieval scores that are generally used for information retrieval tasks. The *K Nearest Neighborhood* (K-NN) algorithm is the first step of the calculation of these scores. Rather than the clustering algorithms, the K-NN algorithm always gives the same results for certain settings of the embedding vectors. Thus, these score values are used in many studies in order to compare the methods in the literature of deep metric learning in a fair manner [26]. In this study, we evaluate

the performances of the models with this type of score due to the same reasons with [26]. These scores are explained in the subsequent subsections.

2.2.1 Recall at K

Recall at K ($R@K$ in short) is the most common metric that the researchers have utilized for evaluating their metric learning algorithms' effectiveness. This evaluation metric does not need any clustering algorithm to assess the score of the embedding space, as explained before. In the literature of information retrieval, this metric has also been called *Recall Rate at K* [23] in order not to distinguish it with recall metric (named as *sensitivity* as well) in the information retrieval area even if they are strictly correlated with each other. The definition of recall for the information retrieval problems is as follows [23]: It is the rate of the intersection of relevant & retrieved items and the relevant ones. In other words, it is calculated by true positive value divided by the sum of true positive and false negative values, as shown in Equation (2.27), where TP , FN stands for true positive and false negatives, respectively.

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|} = \frac{TP}{TP + FN}. \quad (2.27)$$

On the other hand, *recall rate at K* has similar reasoning compared to *Recall* metrics as mentioned in the study of [7]. First, the *top K* recommendations are determined in the embedding space. For this purpose, K-NN items to the anchor sample are used for which the distance is generally defined as ℓ_2 norm. If *any* of the recommended items are in the same class with the anchor item, it accounts as 1; otherwise, it is 0. The overall score is the mean of all items in the dataset, i.e., the score shows how much item has the same label as at least one item in the *top K* recommended item list of itself. The definition of $R@K$ in mathematical notation is given below:

$$R@K = \frac{1}{N} \sum_{n=1}^N r_n(K),$$

$$\text{where } r_n(K) = \begin{cases} 1, & \text{if } \exists i \in \mathbb{Z} \mid (y_n = y_n^i) \wedge (0 < i \leq K), \\ 0, & \text{Otherwise,} \end{cases} \quad (2.28)$$

where y is the label of a certain item, y_n^i is defined as the i^{th} recommended item for the n^{th} item in the dataset, and N is the number of items in the dataset (or test set for the test score).

In addition, there is more than one definition of the evaluation of $R@K$ due to the disagreement on the definition of relevant items. In the book [9], the definition of the $R@K$ is given as follows:

$$R_2@K = \frac{1}{N} \sum_{n=1}^N r_n(K), \quad (2.29)$$

$$\text{where } r_n(K) = \frac{1}{R_{y_n}} \sum_{i=1}^K \delta[y_n^i - y_n],$$

where R_c is defined as number of item that is in the class c minus 1 (due to the fact that the reference item should not be accounted), δ is the Kronecker delta function ($\delta[\alpha - \beta] := 1$ if $\alpha = \beta$; else 0). However, this usage is not common in the area of metric learning.

The algorithms require getting higher scores of $R@K$ in order to show they manipulate the embedding space to be well-clustered. When analyzing the evaluation metric carefully, a drawback can be noticed: The score function does not take into account of the number of *false positives* in the *top K* recommendation list. As a consequence, some studies [26] argue that this evaluation score does not present the exact situation of the embedding space.

2.2.2 Precision at K

The precision is the synonym of *positive predictive value* in the information retrieval literature. It is the ratio of both relevant and retrieved items and the number of retrieved ones, as shown in Equation (2.30). In short, it measures the accuracy of predictions.

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|} = \frac{TP}{TP + FP}. \quad (2.30)$$

Similar to the Equation (2.27), the precision score is calculated by true positives divided by the sum of true positive and false positives, shown in Equation (2.30) as

well, where *FP* stands for false positives.

As before, *Precision at K* (or $P@K$ in short) is similar to (2.30). The ranked list (usually, this list contains the *top-K* retrieved items as mentioned in Section 2.2.1) are utilized in the concept of $P@K$. The score is calculated as shown in Equation (2.31), where $p_i(K)$ is the ratio of relevant item in the i^{th} retrieved item list (the length of the list is K).

$$P@K = \frac{1}{N} \sum_{n=1}^N p_n(K), \quad (2.31)$$

where $p_n(K) = \frac{1}{K} \sum_{i=1}^K \delta[y_n^i - y_n]$.

Similar to the $R@K$, the higher value of $P@K$ implies the better-clustered embedding space. Although the $R@K$ does not reckon with the *false positives*, $P@K$ considers the effect of them. However, this time, it does not attach importance to the ranking order of the false positives. Another deficiency of this evaluation metric is the lack of consideration of the number of relevant items. It means, if the K value in the Equation (2.31) is increased too much, the score cannot present the performance of clustering. Moreover, in the cases where the numbers of relevant items vary depending on each anchor item, the score may mislead.

2.2.3 R-Precision

R-precision (or $P@R$) is also a precision ratio at K , where K is the specific number of each class, R_c which is the same as the one in (2.29). Then, the score function is shown in Equation (2.32), where the $p_n(\cdot)$ is the same as Equation (2.31).

$$P@R = \frac{1}{N} \sum_{n=1}^N p_n(R_{y_n}). \quad (2.32)$$

This metric overcomes one of the drawbacks in the $P@K$, in which the K value of any item is fixed in order not to be faced with the problems resulting from choosing K ,

mentioned in Subsection 2.2.3. On the other hand, it still does not solve the problem owing to the ranking order.

2.2.4 Mean Average Precision at K

Before defining the mean average precision, the average precision should be discussed. The *Average precision* score (or *AveP*) tries to combine both the recall and precision scores for ranked retrieval lists. The formula for calculating the score at K for n^{th} item in the dataset is shown in the Equation (2.33). As it can be observed in the formula, this score evaluation solves the ranking order problem faced in the *Precision@K*.

$$AveP_n@K = \left(\frac{1}{K} \sum_{i=1}^K \delta[y_n^i - y_n] \right) \cdot \delta[y_n^K - y_n]. \quad (2.33)$$

The *mean average precision* (or *MAP*) is defined as the arithmetic mean of *AveP* over all the items in the dataset as shown in Equation (2.34), where N is the number of item in the dataset.

$$MAP@K = \frac{1}{N} \sum_{n=1}^N AveP_n@K. \quad (2.34)$$

In the same vein with Section 2.2.3, *MAP@R* score can be evaluated by Equation (2.35).

$$MAP@R = \frac{1}{N} \sum_{n=1}^N \left(\frac{1}{R_{y_n}} \sum_{i=1}^{R_{y_n}} \delta[y_n^i - y_n] \right) \cdot \delta[y_n^{R_{y_n}} - y_n]. \quad (2.35)$$

2.3 Benchmark Datasets

The datasets on which the models are trained have a quite important role in the area of deep learning. It is because they directly determine whether the parameters in the deep neural networks overfit onto the train set or not while training. In the early stages

of deep learning models such as LeNet [21] in the 1990s, overfitting was the most important issue due to the number of samples in the datasets are too small, whereas the models have too many parameters to train. The order of magnitudes of the number of samples in these datasets is roughly 10^5 [22, 19]. Later on, the bigger datasets can be collected from the Internet, such as ImageNet [5] in the 2010s. ImageNet is the most well-known and one of the biggest datasets utilized for training the deep learning based models. It has over 14 million images, which is significantly much more than the previous datasets. As a matter of fact, this is the reason why AlexNet [20] won the ImageNet image classification competition in 2012: When the number of samples in the dataset increases, the classical models cannot rise their performance rather than the performance increment of the deep learning models as shown in the ImageNet image classification competition in 2012.

There are plenty of datasets utilized for different purposes. To illustrate, some imbalanced ones are used for assessing the robustness of the models against this property of the training data. Another example is that the small datasets (there are only 3 or 5 samples in each class of train set) are exploited in the field of few-shot learning. In the next section, we explain the fine-grained datasets in detail: Their difficulties, the splitting techniques, the most utilized datasets in the literature of deep metric learning.

2.3.1 Definition of Fine-Grained Datasets

The definition of the fine-grained datasets can be defined by their property being only distinguishable with a slight difference of the instances [39]. While standard datasets include more than one meta-category (for example; automobiles, horses are included in the CIFAR-10 [19]), the fine-grained ones consist of only one meta-category and more than one sub-categories as shown in Figure 2.7.

The general task is to classify or cluster these sub-categories rather than the meta ones. For example, the Stanford Cars-196 dataset [17], which is one of the fine-grained datasets, contains only the *car* meta-category, but it has several models of the cars. Apparently, the fine-grained datasets require more effort in order to get acceptable performance with respect to their standard counterparts. To elaborate, the additional challenges for these datasets to the coarse-grained datasets will be explained in the

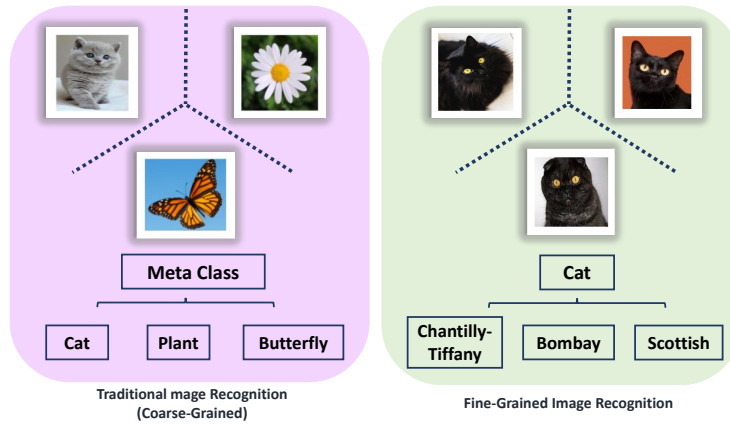


Figure 2.7: Difference between the fine and coarse-grained datasets (figure inspired by [39]).

subsequent subheadings.

2.3.2 Challenges of Fine-Grained Datasets

The challenges for classifying or clustering the fine-grained datasets are divided into 3 main categories by merging the explanations of several surveys on fine-grained datasets in [39, 44, 45, 38]:

2.3.2.1 Sub-Category Challenge



Figure 2.8: The images at the same column of the figure belong to the same types of aircraft, but the visual similarity is smaller in each row. (Figure is inspired by [39]).

The first challenge can be named as *sub-category problem*. This problem comes from

the fact that the classification task for a fine-grained dataset is to distinguish the sub-categories of the *single* meta-category. It creates larger intra-class variance, whereas, the smaller inter-class variance as shown in Figure 2.8. Although the images in the row of Figure 2.8 are similar to each other, the images in the same column are in the same class. Then, the task is to distinguish the images in the same column. This situation reduces the retrieval performance scores since the clusters of the different classes may be open to diffuse with each other.

2.3.2.2 Localization Problem

The second issue is that only a slight and subtle difference in the local regions of the objects should be focused on. It means the background of the images may mislead the model during the decision of its class. For instance, there are 200 types of birds in the CUB-200-2011 dataset [40], and they can be distinguished by the help of beak, wing, tail differences, only. However, the background of the images in the same class may vary. For example, it can be a sky image if the image is taken when the birds are flying or a sea image if the birds are settled. The fact that the photos of the same type of birds can be taken at a different time creates deceptive backgrounds for the models. On the other hand, in the CIFAR-10 dataset, although the background of the airplane class images is usually air, the background of the ship class images are the sea. These can be used as a hint while predicting the label of the images in the coarse-grained datasets. It causes the rigorously-designed loss functions, and networks are needed to reach the high performance scores.

2.3.2.3 Labeling Difficulties

The final problem causes by the relatively small size of the fine-grained datasets compared to the standard ones such as ImageNet. The reason for this issue is due to the fact that it is really difficult to label such datasets. Although anyone can distinguish between a plane and a car from images, only the experts can do it between Artic and Caspian Terns, which are bird types in the CUB-200-2011 [40]. Moreover, The labeling takes more time as compared to the time needed for distinguishing the stan-

standard datasets. Therefore, labeling the fine-grained dataset is costly from both time consumption and the monetary point of view. Consequently, the smaller datasets are usually obtained for fine-grained tasks. The order of magnitudes of these datasets is given in Section 2.3.3, which includes roughly 10^5 samples.

2.3.3 Benchmark Datasets of Deep Metric Learning

Several surveys [26, 15] claim that there are mainly three datasets that are frequently exploited in the deep metric learning studies, and almost all the performance scores of the methods are calculated on these datasets: CUB200-2011 [40], Stanford Cars-196 [17], and Stanford Online Products [33]. The first two of them are also exploited as the part of the FG-comp 2013 challenge, whose website is <https://sites.google.com/site/fgcomp2013/>, but the third one, the Stanford Online Products (SOP) [33], which has 22,634 classes and 120,000 images are in the dataset. Even if SOP is a perfectly balanced dataset, there are only 5 images in each class. Hence, it may create problems related to the few-shot learning topic, which is out of the scope of this thesis. The FGVC Aircraft Dataset [24] can be chosen as the third dataset for analysis of the performance of the models trained with various loss functions. The properties of these datasets are shown in Table 2.1.

Table 2.1: The table of fine-grained benchmark datasets. The number of classes and the total number of images that are included in each dataset are shown in this table, as well as the meta-class of them.

Name	Meta-Class	# Classes	# Images
CUB200-2011 [40]	Birds	200	11,788
Stanford Cars-196 [17]	Cars	196	16,185
FGVC Aircraft [24]	Aircrafts	100	10,000

2.3.4 Different Splitting Techniques of the Datasets

Generalization is one of the most important issues that should be considered while determining how to split the dataset. It is due to the fact that most of the algorithms in machine learning may overfit the training data during optimizing, as mentioned

above. For analyzing the generalization power of the model that is trained on the train set split from the dataset, the performance score of the model should be calculated on the test set that is not seen by the model during the training phase. Therefore, the standard way to split the dataset in machine learning forces that the samples in the train and test sets are disjoint [10]. Moreover, the lecture notes in [10] suggests splitting the train set into 2 parts: Train and validation sets. Their images are also disjoint in order to adjust the hyper-parameters and the early stopping. This type of splitting is called *close-set* settings as shown in Figure 2.9a.

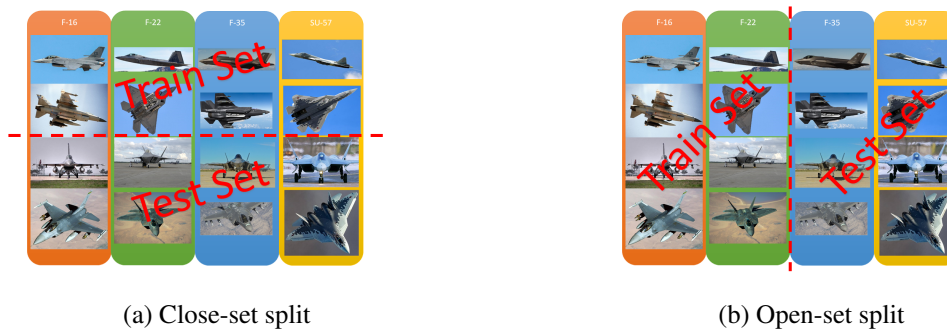


Figure 2.9: The train and test sets of different types of splits for the CUB-200-2011 [40] dataset as shown in Figure 2.8.

Another type of splitting is the *open-set* split, also called as *zero-shot learning* scenario as in Figure 2.9b. In this scenario, in addition to the disjoint samples in the train and test sets, the classes of these sets are also disjoint. This case means there is no sample in the test set that is in the same class as any sample in the train set. In this case, the model should also learn the *features* rather than learning only the class probabilities. The model may be more robust to variation of the samples in the dataset and overfitting the classes in the dataset. Indeed, although it gives lower scores on the test set, it may be more useful in real-world applications. In other words, the trained model will be not specific for the *data* but specific for the *task*.

CHAPTER 3

DISTANCE SENSITIVE ENTANGLED LOSS FUNCTION

Deep metric learning studies in the literature focus on loss function design and mining methods. The researchers, who studied in this field, usually explain their motivation over how much *penalty* the loss function should evaluate for the situations they have designed.

In this section, we focus on the loss functions in the literature and propose novel loss functions. There are various loss functions that have already been proposed. In recent years, some studies [26] argue that these loss functions might reach quite similar performance scores; i.e., they do not differ from each other.

In this study, we analyze and try to explain the reason behind this observation. During this analysis, we should reduce the multi-dimensional embedding space to \mathbb{R}^2 space in order to visualize the results. At this point, we exploit some natural phenomena to discover an optimal loss function.

Before starting the analysis, we would like to make a weak analogy between physical forces that determines the movement of particles and the feature vectors in the embedding space in Section 3.1. After determining the distilled form of the loss functions (ones without hinge function, which intrinsically leads mining the samples in the embedding space as explained in Section 2.1.1), the hyper-parameters of these loss functions, their effects, and the ideal hyper-parameters for the certain training settings are discussed in Section 3.2.

3.1 A Weak Analogy between Feature Vectors and Point Charges

One can argue that there is an interesting analogy between the feature vectors in the embedding space and some point electrical charges in free space. Nevertheless, while electrostatic force pushes objects with the same charges, the same force also attracts those in the opposite charges, whose magnitude is given in the relation below, as *Coulomb's Law*:

$$\|F\| = \frac{q_1 \cdot q_2}{4\pi\epsilon_r d^2}. \quad (3.1)$$

Being a *weak* analogy, this physical phenomena creates an *opposite* situation to our objective in deep metric learning, which requires similar objects pulling each other in the embedding space into the nearby locations, whereas the objects from different class pushing themselves far away from each other. Nevertheless, once we re-examine Coulomb's law for electrostatic force in Equation 3.1, one can state that the scenarios, in which the *permittivity constant* (ϵ_r) is negative (which might happen in real-life for some certain frequencies in some *meta-materials*); hence, the well-known physical phenomena still become analogous to our problem for some materials. At this point, let us assume ϵ_r be equal to a negative constant in the static case.

Assume that there are two positive and one negative charged point charge in free space, as shown in Figure 3.1. Under these conditions, the mathematical relation for the motion of these point charges can be obtained by the governing equations of classical mechanics. Indeed, the objects move according to the solution of Euler-Lagrange equations of motion [34], as given in below:

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = 0, \quad (3.2)$$

where q_i 's are the generalized coordinates, t denotes the time, and L is the Lagrangian depending on the generalized coordinates, their derivatives and time ($L = L(q, \dot{q}, t)$). However, the Lagrangian, in this case, contains several non-linear terms, which makes Equation (3.2) hard to solve. Therefore, one can iteratively find out where the particles should be located in the next time step through the evaluation of force vectors by Newtonian mechanics.

This process requires the calculation of the forces repeatedly at each time interval. In these calculations, all of the point charges interact with each other. It is assumed that

there is no extra external force, and all of the magnetic effects due to the motion of these point charges are ignored for simplicity.

Every interaction affects both 2 point charges as a force. These forces are equal to each other, but in opposite directions, according to the 3rd law of Newton. Moreover, the total force affecting each object is the superposition (vector sum) of each force affecting it. By using the equation of the Coulomb's Law shown in (3.1) where $\epsilon_r < 0$, one can determine the directions of the forces as shown in Figure 3.1.

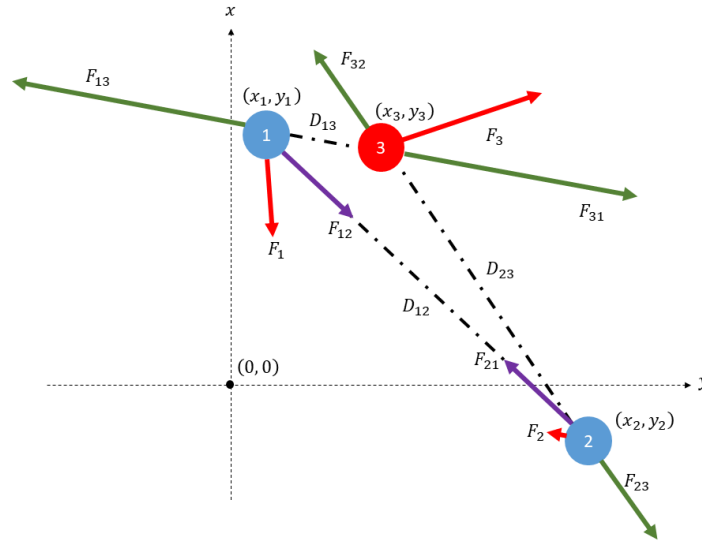


Figure 3.1: In this example illustration, it is assumed that all particles have the same amount of electrical charge. While Objects #1 & #2 have the same sign, and Object #3 has a reverse sign. Hence, the first two attract each other and repel the third one, when $\epsilon_r < 0$. The interaction between particles i and j is shown as the force denoted by F_{ij} . The resultant force exerted on particle i is calculated by the vector summation ($F_i = \sum_{\forall j} F_{ij}$). Each particle moves in the direction of its resultant electrical force vector. The location of the object #1, #2, and #3 are at (x_1, y_1) , (x_2, y_2) , and (x_3, y_3) , respectively.

Coulomb's Law also gives information about the magnitudes of these vectors. The magnitude of the electrostatic force between the two point charges changes proportionally to the square of the reciprocal of the distance between these points, as dictated by (3.1). Then, the particles move with the least displacement that will allow the potential energy of the system to decrease. However, in deep metric learning, the

performance score (which are explained in Section 2.2) might differ from the potential energy analogy, which *typically* changes with $1/d$ in contrast to the inverse-square Law of attraction [4]. Hence, this change can also be considered as a design parameter while proposing a loss function.

We conclude this section by defining the following two functions, $g_{pull}(\cdot)$ and $g_{push}(\cdot)$. To elaborate, we define $\|F_{12}\| = \|F_{21}\| = g_{pull}(D_{12})$, whereas $\|F_{13}\| = \|F_{31}\| = g_{push}(D_{13})$, for which D_{12} and D_{13} are the Euclidean distance between Objects #1 & #2, and #1 & #3, respectively, where $D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ for 2-dimensional case. This situation is discussed in detail in Section 3.1.3.

A final point to emphasize is the optimizers in deep metric learning. Iterative optimization methods are frequently used during the training of neural networks. For example, one of the most primitive optimizers used to train a neural network is stochastic gradient descent, and it basically uses the steepest decent formula for the optimization problem $\hat{x} = \arg \min_x \mathcal{L}(x)$, as shown below:

$$x_{n+1} = x_n - \eta \cdot \nabla_x (\mathcal{L}(x_n)). \quad (3.3)$$

The update rule in this formula varies in terms of both magnitude and direction depending on the first derivative of the model, denoted as $\nabla_x (\mathcal{L}(x_n))$.

At this point, there is an analogy between the loss function \mathcal{L} of metric learning and *work* (or potential energy). To elaborate, the derivative of work with respect to displacement gives the force acting on the object. Similarly, when we use an optimizer that only needs the 1st derivative of the loss to train the parameters of the model, the new update will now depend on the first derivative of the loss with respect to the embedding vectors, which can be thought as the coordinates of the samples in the embedding space, similar to the location of the object in free space. Hence, one can regard the term $\nabla_x (\mathcal{L}(x_n))$ as a force. The time interval also changes with the size of the learning rate, denoted as η .

Before defining a novel loss function, most of the loss functions available in the literature are examined from this perspective. Using our analogy, we calculated the so called *forces* of these losses and showed them in Tables 3.1 and 3.2, which are the same as the gradients of the loss functions explained in Section 2.1.1. Note that the

hinge loss ($\mathcal{L}_{hinge}(\cdot) = \max(0, \cdot)$, also named as *ramp function*) only eliminates some of the triplets but does not affect the direction and the magnitude of the partial derivative of the losses (or so-called forces) in these tables. Thus, the partial derivatives are calculated as if there are no hinge losses in the loss functions as also mentioned in Section 2.1.1.2.

Mathematically speaking, the partial derivatives of the original loss functions are $\frac{\partial \mathcal{L}}{\partial f_i} = \frac{\partial \mathcal{L}_{hinge}(\mathcal{L}_{in})}{\partial \mathcal{L}_{in}} \cdot \frac{\partial \mathcal{L}_{in}}{\partial f_i}$, where the original loss is $\mathcal{L} = \max(0, \mathcal{L}_{in})$, and f_i is the embedding vector of i^{th} instance (\mathcal{L}_{in} functions are given in these tables). The term $\frac{\partial \mathcal{L}_{hinge}(\mathcal{L}_{in})}{\partial \mathcal{L}_{in}}$ equals to Heaviside step function (also named as *unit step function*). That is, it filters out the triplets when $\mathcal{L}_{in} < 0$, and gives 1 when $\mathcal{L}_{in} > 0$. Moreover, the case where $m \gg \|f_i - f_j\|_2$ for any $x_i, x_j \in D$ the \mathcal{L}_{in} value cannot be negative; therefore, the equality $\frac{\partial \mathcal{L}_{hinge}(\mathcal{L}_{in})}{\partial \mathcal{L}_{in}} = 1$ always holds.

3.1.1 Derivation of the Novel Loss Function

At this point, we implement our loss function, inspired by the aforementioned electrostatic forces. However, unlike most of the design patterns encountered in the literature, in our approach, we set the forces between samples first. The forces in our analogy depend on the location of the objects. The location of an object in free space is equivalent to the coordinate of a sample in the embedding space, i.e., the feature vector extracted from its image.

Since there are three samples in the triplet, we consider that each of them interacted with the other two; hence, we are able to write the equations for six forces. We have set the magnitudes of these six forces as constants in the first stage. In this way, we write the partial derivatives of the loss function as below:

$$\begin{aligned}\frac{\partial \mathcal{L}_{generic}}{\partial f_a} &= c_1 \cdot (f_n - f_a) + c_2 \cdot (f_a - f_p), \\ \frac{\partial \mathcal{L}_{generic}}{\partial f_p} &= c_3 \cdot (f_n - f_p) + c_4 \cdot (f_p - f_a), \\ \frac{\partial \mathcal{L}_{generic}}{\partial f_n} &= c_5 \cdot (f_a - f_n) + c_6 \cdot (f_p - f_n).\end{aligned}\tag{3.4}$$

In the equation above, c_i parameters are constants. For merging the c_i 's in (3.4) and the forces shown in Figure 3.1, the equalities in Equation (3.5) can be written when

the anchor, positive, and negative instances in a triplet sample are corresponding to the objects #1, #2, and #3, respectively.

$$\begin{aligned} c_1 &= \frac{\|F_{13}\|}{\|f_a - f_n\|_2}, & c_2 &= \frac{\|F_{12}\|}{\|f_a - f_p\|_2}, & c_3 &= \frac{\|F_{23}\|}{\|f_p - f_n\|_2}, \\ c_4 &= \frac{\|F_{21}\|}{\|f_a - f_p\|_2}, & c_5 &= \frac{\|F_{31}\|}{\|f_a - f_n\|_2}, & c_6 &= \frac{\|F_{32}\|}{\|f_p - f_n\|_2}. \end{aligned} \quad (3.5)$$

Once we take the integrals of these equations, one finds the loss functions, which are dependent on the $h_i(\cdot, \cdot)$ functions, shown in Equation (3.6).

$$\begin{aligned} \mathcal{L}_{generic} &= f_a^T (c_1 f_n - c_2 f_p) + \frac{c_2 - c_1}{2} \|f_a\|_2^2 + h_1(f_p, f_n), \\ &= f_p^T (c_3 f_n - c_4 f_a) + \frac{c_4 - c_3}{2} \|f_p\|_2^2 + h_2(f_a, f_n), \\ &= f_n^T (c_5 f_a + c_6 f_p) - \frac{c_5 + c_6}{2} \|f_n\|_2^2 + h_3(f_a, f_p). \end{aligned} \quad (3.6)$$

It should be noted that it is easy to show the equivalence between (3.6) and (3.4) by taking the partial derivatives of (3.6) with respect to f_a , f_p and f_n .

In order to solve these three equations in a coupled way, three constraints $c_1 = c_5$, $c_2 = c_4$, $c_3 = c_6$ must be provided. If these statements hold, we can combine the expressions and get the general loss function as below:

$$\begin{aligned} \mathcal{L}_{generic}(\tau) &= c_1 f_a^T f_n - c_2 f_a^T f_p + c_3 f_p^T f_n + m \\ &\quad + \frac{1}{2} \left((c_2 - c_1) \|f_a\|_2^2 + (c_2 - c_3) \|f_p\|_2^2 - (c_1 + c_3) \|f_n\|_2^2 \right), \end{aligned} \quad (3.7)$$

where m is a *margin* (constant) and $\tau = \{f_a, f_p, f_n\}$ is a triplet sample.

In Equation (3.7), the distances are defined in ℓ_2 norm (similar results can be obtained when the distance is defined as cosine similarity). In this way, a similarity is established between the loss function in deep metric learning and electrostatic forces.

The loss functions will be examined for equilateral and non-equilateral triangle cases in the following sections to understand their behavior better in these examples.

3.1.2 Special Case #1: Equilateral Triangle

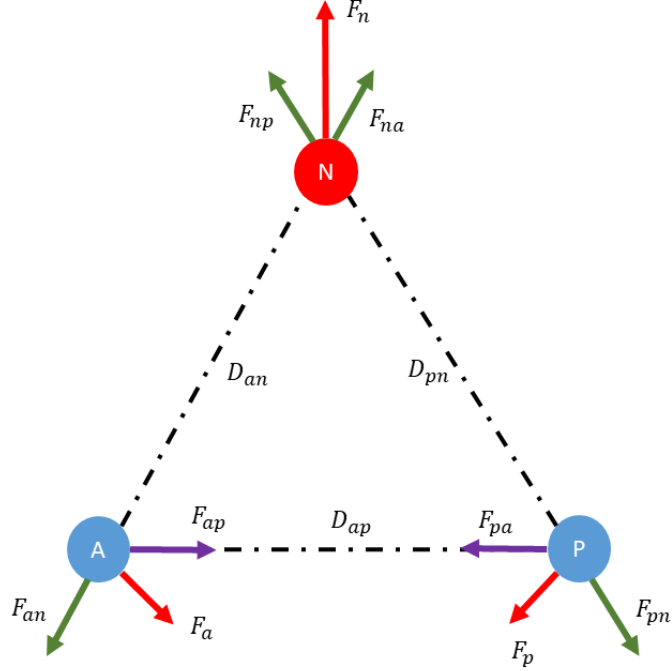


Figure 3.2: The 2-dimensional representation of embedding space in equilateral triangle case. The equality $D_{an} = D_{ap} = D_{pn} = 1$ is hold. The objects A and P are belonging to the same class, whereas the classes of the object N is different from them.

As seen in Figure 3.2, let the distances be the same between all the samples to each other in the embedding space and equal to the unit distance. In this case, ρ can be defined as a ratio between pull and push forces ($\rho = g_{push}(d)/g_{pull}(d), \forall d \in \mathbb{R}^{\geq 0}$). If we choose ρ to be 1; then, $c_1 = c_2 = c_3$ in Equation (3.7). For the sake of simplicity, let the magnitudes of all the forces are equal to 1 ($\rho \cdot \|F_{ap}\| = \|F_{an}\| = \|F_{pn}\| = 1$); hence, the c_i 's are equal to 1. The loss function in Equation (3.8) can be obtained, which we denoted it as *entangle loss function*.

$$\begin{aligned} \mathcal{L}_{entangle}(\tau) &= f_a^T f_n - f_a^T f_p + f_p^T f_n + \|f_n\|_2^2 + m, \\ &= \frac{1}{2} \|f_a - f_p\|_2^2 - \frac{1}{2} \|f_a - f_n\|_2^2 - \frac{1}{2} \|f_p - f_n\|_2^2 + m. \end{aligned} \quad (3.8)$$

When we take the partial derivatives with respect to f_a , f_p , and f_n to verify it, we get the equalities in Equation (3.9).

$$\begin{aligned}
\frac{\partial \mathcal{L}_{entangle}}{\partial f_a} &= (f_n - f_a) + (f_a - f_p), \\
\frac{\partial \mathcal{L}_{entangle}}{\partial f_p} &= (f_n - f_p) + (f_p - f_a), \\
\frac{\partial \mathcal{L}_{entangle}}{\partial f_n} &= (f_a - f_n) + (f_p - f_n).
\end{aligned} \tag{3.9}$$

In similar reasoning, another loss can be proposed, as *location aware loss function*, that is inspired by the entangle loss. The partial derivatives of this loss function are given in Equation (3.10). We simply add their feature vectors in the partial derivatives of themselves.

$$\begin{aligned}
\frac{\partial \mathcal{L}_{location}}{\partial f_a} &= f_a - f_p + f_n, \\
\frac{\partial \mathcal{L}_{location}}{\partial f_p} &= f_p - f_a + f_n, \\
\frac{\partial \mathcal{L}_{location}}{\partial f_n} &= f_n + f_a + f_p.
\end{aligned} \tag{3.10}$$

The integral of the terms in Equation (3.10) is shown in Equation (3.11).

$$\begin{aligned}
\mathcal{L}_{location} &= f_a^T (f_n - f_p) + \frac{1}{2} \|f_a\|_2^2 + h_1(f_p, f_n), \\
&= f_p^T (f_n - f_a) + \frac{1}{2} \|f_p\|_2^2 + h_2(f_a, f_n), \\
&= f_n^T (f_a + f_p) - \frac{1}{2} \|f_n\|_2^2 + h_3(f_a, f_p).
\end{aligned} \tag{3.11}$$

By using this loss function, we try to analyze whether the performance score is increased when the forces depend on the location of the instances. The loss function obtained by the similar operations performed is shown in Equation (3.12).

$$\mathcal{L}_{location} = \frac{1}{2} (\|f_a - f_p\|_2^2 - \|f_a - f_n\|_2^2 + \|f_a\|_2^2 + 2f_p^T f_n + 2\|f_n\|_2^2) + m. \tag{3.12}$$

3.1.3 Special Case #2: Non-Equilateral Triangle

This situation is more challenging compared to the first one. In this case, we can change the distances between the instances being not equal to the unit distance as shown in Figure 3.3. Now, the $g_{pull}(\cdot)$ and $g_{push}(\cdot)$ functions gain importance, since the forces in the triplet samples vary depending on these functions.

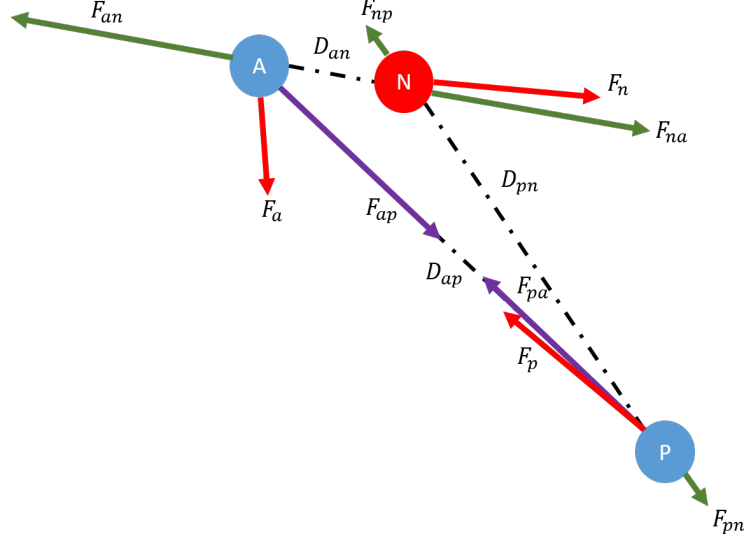


Figure 3.3: The 2-dimensional representation of feature space in the non-equilateral triangle case. Note that the inequalities $D_{ap} \neq D_{an}, D_{ap} \neq D_{pn}, D_{an} \neq D_{pn}$ are hold.

The first suggested loss is based on the entangle loss. As before, the pull and push functions are taken as constants, i.e., $g_{push}(d) = \rho \cdot g_{pull}(d) = \rho, \forall d \in \mathfrak{R}^{\geq 0}$. For this case, let c_i 's be $1/\|f_j - f_k\|_2$ for the same class instances, or $\rho/\|f_j - f_k\|_2$ if they are in the different classes for certain j and k , whose the partial derivatives are shown below:

$$\begin{aligned}
 \frac{\partial \mathcal{L}_{modified}}{\partial f_a} &= \rho \cdot \frac{f_n - f_a}{\|f_a - f_n\|_2} + \frac{f_a - f_p}{\|f_a - f_p\|_2}, \\
 \frac{\partial \mathcal{L}_{modified}}{\partial f_p} &= \rho \cdot \frac{f_n - f_p}{\|f_p - f_n\|_2} + \frac{f_p - f_a}{\|f_a - f_p\|_2}, \\
 \frac{\partial \mathcal{L}_{modified}}{\partial f_n} &= \rho \cdot \left(\frac{f_a - f_n}{\|f_a - f_n\|_2} + \frac{f_p - f_n}{\|f_p - f_n\|_2} \right).
 \end{aligned} \tag{3.13}$$

Closed form integrals of the relations in Equation (3.9) are taken. (Note that the statement $\frac{\partial}{\partial x} \|x\|^n = n \cdot x \cdot \|x\|^{n-2}$ holds.) The new loss function which covers

more cases is obtained in Equation (3.14), which is named as *modified entangle loss function*.

$$\mathcal{L}_{modified}(\tau) = \|f_a - f_p\|_2 - \rho \cdot (\|f_a - f_n\|_2 + \|f_p - f_n\|_2) + m. \quad (3.14)$$

Obviously, similar losses can be obtained by changing ρ . In fact, the ρ value should be regarded as a hyper-parameter that is tuned during training with the help of the feedback coming from the validation score.

3.1.4 Losses with Non-Constant Pull and Push Functions

Another loss function proposal could be based upon non-constant pull and push functions. In order to design these functions, first, we should focus on the objective of training the model.

The main aim is to increase the performance by using a fewer number of iterations since the model parameters may be stuck in a local minimum or a saddle point during these iterations, which might reduce the performance score. Moreover, the retrieval scores are often used for evaluating the performance of the model.

The retrieval scores are basically calculated with the help of K nearest neighbors. In other words, these scores are calculated based on the similarity between the labels of the K instances closest to a selected reference instance. In order to increase the retrieval scores, we should attach our importance to the K nearest instances by incorporating distances into the loss function. Specifically, the task of the function g_{push} is to disperse the instances in the different classes. This force is similar to the electrostatic force; the magnitude of the force is smaller when the objects are far from each other. The reason is due to the fact that the distant negatives are desired and should not update the anchor's location significantly. On the other hand, the close negatives should be moved far apart, since they reduce the performance score significantly. Then, one can conclude that the $g_{push}(\cdot)$ function should be monotonically decreasing, i.e. $g_{push}(x) \geq g_{push}(y)$ where $x \leq y, \forall x, y \in \mathfrak{R}^{\geq 0}$.

Similarly, the samples of the same class that are close to each other do not reduce

the performance. More importantly, the performance score is not related to the closeness of the samples. This score just concerns if these objects are the closest ones or not. Conversely, the performance score decreases dramatically when any positive and anchor instances are far from each other. Therefore, the penalty of this situation should be high in order to get them closer locations with a fewer number of iterations. As a result, the $g_{pull}(\cdot)$ function should be monotonically increasing, i.e. $g_{pull}(x) \leq g_{pull}(y)$ where $x \leq y, \forall x, y \in \mathfrak{R}^{\geq 0}$.

In order to propose a novel loss function that meets the above conditions, we revise the partial derivatives in Equation (3.4) into Equation (3.15) by setting $g_{push}(d) = \rho/d^r$ and $g_{pull}(d) = d^s$. In Equation (3.15), the values of r and s are positive integers. It should be noted that the case, where $r = -1, s = 1$ and $\rho = 1$, is expressed as *entangle loss function* in Equation (3.8), and the case, for which $r = 0$ and $s = 0$, is named as *modified entangle loss function* in Equation (3.14).

$$\begin{aligned}\frac{\partial \mathcal{L}_{distance}}{\partial f_a} &= \rho \cdot \frac{f_n - f_a}{\|f_a - f_n\|_2^{r+1}} + (f_a - f_p) \cdot \|f_a - f_p\|_2^{s-1}, \\ \frac{\partial \mathcal{L}_{distance}}{\partial f_p} &= \rho \cdot \frac{f_n - f_p}{\|f_p - f_n\|_2^{r+1}} + (f_p - f_a) \cdot \|f_a - f_p\|_2^{s-1}, \\ \frac{\partial \mathcal{L}_{distance}}{\partial f_n} &= \rho \cdot \left(\frac{f_a - f_n}{\|f_a - f_n\|_2^{r+1}} + \frac{f_p - f_n}{\|f_p - f_n\|_2^{r+1}} \right).\end{aligned}\quad (3.15)$$

When we take the integrals of the partial derivatives in Equation (3.15), we get Equation (3.16) where $s \neq -1$ and $r \neq 1$.

$$\begin{aligned}\mathcal{L}_{distance} &= \frac{\|f_a - f_p\|_2^{s+1}}{s+1} - \rho \cdot \frac{\|f_a - f_n\|_2^{1-r}}{1-r} + h_1(f_p, f_n), \\ &= \frac{\|f_a - f_p\|_2^{s+1}}{s+1} - \rho \cdot \frac{\|f_p - f_n\|_2^{1-r}}{1-r} + h_2(f_a, f_n), \\ &= -\frac{\rho}{1-r} \cdot (\|f_a - f_n\|_2^{1-r} + \|f_p - f_n\|_2^{1-r}) + h_3(f_a, f_p).\end{aligned}\quad (3.16)$$

Hence, we can write the *distance sensitive loss function* as below:

$$\mathcal{L}_{distance} = \frac{\|f_a - f_p\|_2^{s+1}}{s+1} - \frac{\rho}{1-r} \cdot (\|f_a - f_n\|_2^{1-r} + \|f_p - f_n\|_2^{1-r}) + m. \quad (3.17)$$

There are four hyper-parameters that determine how the loss function affects the embedding space in (3.17). These hyper-parameters are discussed in subsequent subsections.

Before discussing the hyper-parameters and the hinge loss added version of the proposed loss, we try to show the significant difference between the distance sensitive & location-aware loss functions and the losses in the literature briefly. First, all the triplet-based losses are summarized in Tables 3.1, 3.2 as derived in Section 2.1.1.2. One can observe that the update terms of the losses in the literature are quite similar to each other. This similarity is possibly the reason for having similar evaluation scores in the test sets, which are reported in the study [26]. After these tables, our proposed losses and their gradients are tabulated in Table 3.3.

Table 3.1: The table of the forces of all the triplet losses found in literature in chronological order (from 2014 to 2015).

Loss Name	Loss Function(\mathcal{L})	Derivatives of \mathcal{L} ($\partial\mathcal{L}/\partial f_i$)
Original Triplet Loss [14]	$\left(\frac{e^{\ f_a - f_p\ _2}}{e^{\ f_a - f_p\ _2} + e^{\ f_a - f_n\ _2}}\right)^2$	$\frac{\partial\mathcal{L}}{\partial f_a} = \frac{2 \cdot e^{2\ f_a - f_p\ _2 + \ f_a - f_n\ _2}}{(e^{\ f_a - f_p\ _2} + e^{\ f_a - f_n\ _2})^2} \cdot \left(\frac{(f_a - f_p)}{\ f_a - f_p\ _2} - \frac{f_a - f_n}{\ f_a - f_n\ _2}\right)$ $\frac{\partial\mathcal{L}}{\partial f_p} = \frac{2 \cdot e^{2\ f_a - f_p\ _2 + \ f_a - f_n\ _2}}{(e^{\ f_a - f_p\ _2} + e^{\ f_a - f_n\ _2})^2} \cdot \frac{(f_p - f_a)}{\ f_a - f_p\ _2}$ $\frac{\partial\mathcal{L}}{\partial f_n} = \frac{2 \cdot e^{2\ f_a - f_p\ _2 + \ f_a - f_n\ _2}}{(e^{\ f_a - f_p\ _2} + e^{\ f_a - f_n\ _2})^2} \cdot \frac{f_a - f_n}{\ f_a - f_n\ _2}$
Ranking Loss [36]	$\ f_a - f_p\ _2 - \ f_a - f_n\ _2 + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = \frac{f_a - f_p}{\ f_a - f_p\ _2} - \frac{f_a - f_n}{\ f_a - f_n\ _2}$ $\frac{\partial\mathcal{L}}{\partial f_p} = \frac{f_p - f_a}{\ f_a - f_p\ _2}$ $\frac{\partial\mathcal{L}}{\partial f_n} = \frac{f_a - f_n}{\ f_a - f_n\ _2}$
Ratio Loss [41]	$1 - \frac{\ f_a - f_n\ _2}{\ f_a - f_p\ _2 + m}$	$\frac{\partial\mathcal{L}}{\partial f_a} = \frac{\frac{\ f_a - f_n\ \cdot (f_a - f_p)}{\ f_a - f_p\ _2} - \frac{\ f_a - f_p\ _2 \cdot (f_a - f_n)}{\ f_a - f_n\ _2}}{\ f_a - f_p\ _2 + m}$ $\frac{\partial\mathcal{L}}{\partial f_p} = \frac{\ f_p - f_n\ _2 \cdot (f_p - f_a)}{\ f_a - f_p\ _2 \cdot (\ f_a - f_p\ _2 + m)}$ $\frac{\partial\mathcal{L}}{\partial f_n} = \frac{f_a - f_n}{\ f_a - f_n\ _2 \cdot (\ f_a - f_p\ _2 + m)}$
FaceNet Loss [28]	$\ f_a - f_p\ _2^2 - \ f_a - f_n\ _2^2 + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = f_n - f_p$ $\frac{\partial\mathcal{L}}{\partial f_p} = f_p - f_a$ $\frac{\partial\mathcal{L}}{\partial f_n} = f_a - f_n$

Table 3.2: The table of the forces of all the triplet losses found in literature in chronological order (from 2015 to 2020).

Loss Name	Loss Function (\mathcal{L})	Derivatives of \mathcal{L} ($\partial\mathcal{L}/\partial f_i$)
N-Pairs [32]	$\log(1 + e^{f_a f_n - f_a f_p})$	$\frac{\partial\mathcal{L}}{\partial f_a} = \frac{e^{f_a f_n - f_a f_p}}{1 + e^{f_a f_n - f_a f_p}} \cdot (f_n - f_p)$ $\frac{\partial\mathcal{L}}{\partial f_p} = -\frac{e^{f_a f_n - f_a f_p}}{1 + e^{f_a f_n - f_a f_p}} \cdot f_p$ $\frac{\partial\mathcal{L}}{\partial f_n} = \frac{e^{f_a f_n - f_a f_p}}{1 + e^{f_a f_n - f_a f_p}} \cdot f_a$
Angular Loss [37]	$\ f_a - f_p\ _2^2 - 4 \tan^2(\rho) \cdot \ f_n - \frac{f_a + f_p}{2}\ _2^2$	$\frac{\partial\mathcal{L}}{\partial f_a} = 2 \cdot (f_a - f_p) - 2 \tan^2(\rho) \cdot (f_a + f_p - 2f_n)$ $\frac{\partial\mathcal{L}}{\partial f_p} = 2 \cdot (f_p - f_a) - 4 \tan^2(\rho) \cdot (f_a + f_p - 2f_n)$ $\frac{\partial\mathcal{L}}{\partial f_n} = 4 \tan^2(\rho) \cdot (f_a + f_p - 2f_n)$
Moving Loss [3]	$\ f_a - f_p\ _2^2 - \ f_a - f_n\ _2^2 - \rho \cdot \frac{1 - f_p^T f_a}{\ f_a - f_p\ _2 \cdot \ f_a - f_n\ _2} + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = 2 \cdot (f_n - f_p) - \rho \cdot \left(\frac{f_a - f_p}{\ f_a - f_p\ _2 \cdot \ f_a - f_n\ _2} + \frac{\ f_a - f_p\ _2}{\ f_a - f_n\ _2^3} \cdot (f_n - f_a) \right)$ $\frac{\partial\mathcal{L}}{\partial f_p} = 2 \cdot (f_p - f_a) - \rho \cdot \frac{f_p + f_a}{\ f_a - f_p\ _2 \cdot \ f_a - f_n\ _2}$ $\frac{\partial\mathcal{L}}{\partial f_n} = 2 \cdot (f_a - f_n) - \rho \cdot \frac{f_a + f_n}{\ f_a - f_p\ _2 \cdot \ f_a - f_n\ _2^3}$

Table 3.3: The table of the forces of the proposed triplet losses.

Loss Name	Loss Function (\mathcal{L})	Derivatives of \mathcal{L} ($\partial\mathcal{L}/\partial f_i$)
Entangle	$f_a^T f_n - f_a^T f_p + f_p^T f_n + \ f_n\ _2^2 + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = (f_n - f_a) + (f_a - f_p)$ $\frac{\partial\mathcal{L}}{\partial f_p} = (f_n - f_p) + (f_p - f_a)$ $\frac{\partial\mathcal{L}}{\partial f_n} = (f_a - f_n) + (f_p - f_n)$
Location Aware	$\frac{1}{2} (\ f_a - f_p\ _2^2 - \ f_a - f_n\ _2^2 + \ f_a\ _2^2 + 2f_p^T f_n + 2\ f_n\ _2^2) + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = f_a - f_p + f_n$ $\frac{\partial\mathcal{L}}{\partial f_p} = f_p - f_a + f_n$ $\frac{\partial\mathcal{L}}{\partial f_n} = f_n + f_a + f_p$
Modified Entangle	$\ f_a - f_p\ _2 - \rho \cdot (\ f_a - f_n\ _2 + \ f_p - f_n\ _2) + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = \rho \cdot \frac{f_n - f_a}{\ f_a - f_n\ _2} + \frac{f_a - f_p}{\ f_a - f_p\ _2}$ $\frac{\partial\mathcal{L}}{\partial f_p} = \rho \cdot \frac{f_n - f_p}{\ f_p - f_n\ _2} + \frac{f_p - f_a}{\ f_a - f_p\ _2}$ $\frac{\partial\mathcal{L}}{\partial f_n} = \rho \cdot \left(\frac{f_a - f_n}{\ f_a - f_n\ _2} + \frac{f_p - f_n}{\ f_p - f_n\ _2} \right)$
Distance Sensitive	$\frac{\ f_a - f_p\ _2^{s+1}}{s+1} - \frac{\rho}{1-r} \cdot (\ f_a - f_n\ _2^{1-r} + \ f_p - f_n\ _2^{1-r}) + m$	$\frac{\partial\mathcal{L}}{\partial f_a} = \rho \cdot \frac{f_n - f_a}{\ f_a - f_n\ _2^{r+1}} + (f_a - f_p) \cdot \ f_a - f_p\ _2^{s-1}$ $\frac{\partial\mathcal{L}}{\partial f_p} = \rho \cdot \frac{f_n - f_p}{\ f_p - f_n\ _2^{r+1}} + (f_p - f_a) \cdot \ f_a - f_p\ _2^{s-1}$ $\frac{\partial\mathcal{L}}{\partial f_n} = \rho \cdot \left(\frac{f_a - f_n}{\ f_a - f_n\ _2^{r+1}} + \frac{f_p - f_n}{\ f_p - f_n\ _2^{r+1}} \right)$

3.2 Hyper-Parameters of the Proposed Loss Function

After deriving the distance sensitive loss function, we can add an *in-loss-miner* to this function by including a hinge function, similar to the proposed losses in the literature of metric learning. To define the concept in-loss-miner, it has the same functionality as an online miner. That is, let us select an anchor and a positive instance of the triplet, which are located in fixed points in the embedding space. Possibly there are plenty of negative instances that can form a valid triplet, and what the in-loss-miner does is that it selects only the particular negatives that are located in the region determined by the loss function as explained in Section 2.1.2 for online miners. In order to provide the functionality of in-loss-miner, the hinge function is utilized for filtering out some of the negatives which are located outside of the region, which was determined by the loss function itself. The aim of the utilization of the in-loss-miner is to make our model more robust for the outliers. The generic form of our loss function becomes as follows:

$$\begin{aligned}\mathcal{L}_{distance}^{hinged} &= \min(m_2, \max(0, \mathcal{L}_{distance} + m_1)), \\ \mathcal{L}_{distance} &= \frac{\|f_a - f_p\|_2^{s+1}}{s+1} - \frac{\rho}{1-r} \cdot (\|f_a - f_n\|_2^{1-r} + \|f_p - f_n\|_2^{1-r}).\end{aligned}\tag{3.18}$$

The loss value is clamped from upper and lower bounds by the constants denoted as m_1 and m_2 . To visualize, the relationship between $\mathcal{L}_{distance}^{hinged}$ and $\mathcal{L}_{distance}$ is similar to the transfer characteristics of an op-amp. Then, the loss function in Equation (3.18) generates non-zero gradients when its value is in between these boundaries, i.e. $\frac{\partial \mathcal{L}_{distance}^{hinged}}{\partial \mathcal{L}_{distance}} = 1$ when $-m_1 < \mathcal{L}_{distance} < m_2 - m_1$ holds; otherwise the loss returns 0 value. The analogy with op-amps can be regarded as the generated non-zero gradient region corresponds to the linear region of the op-amp. Note that the constraint $m_2 > 0$ should be provided so that the region of non-zero gradients becomes non-trivial.

The aim for using m_1 is to determine the hyper-volume of the upper bound of the mined region. The larger m_1 causes the larger region for negative samples. On the other hand, the m_2 affects the size (hyper-volume) of the inner region that is not utilized by the model due to the zero gradients generated from the loss function for preventing the model from the large noise in the gradients explained in [42].

Obviously, the values of m_1 and m_2 depend on the hyper-parameters denoted as ρ , s ,

and r . Hence, we discuss how to select these parameters according to the settings of the training procedure.

3.2.1 Discussion on the Ratio between Pull and Push Forces

In this subsection, we try to find the most effective ratio between pull and push forces, which denoted as ρ in Equation 3.18. This ratio highly depends on the number of instances in the dataset or in a batch when the mini-batch based optimization algorithm is utilized. Hence, first, we should draw the picture for a batch update. Let the batch size be 2^x , and 2^y different classes are contained in a batch, where $x, y \in \mathbb{Z}^+$ and $x > y > 0$. (Note that choosing the batch size as the power of 2 is quite efficient for the popular platforms such as PyTorch [27] in practice.) Therefore, there are 2^{x-y} instances for each class in a batch. It is quite apparent that we can find $2^{2x-y} \cdot (2^{x-y} - 1) \cdot (2^y - 1)$ valid triplets in each batch.

For further analysis, there are $2^x - 2^{x-y}$ valid triplets containing certain anchor and positive instances in the batch. A similar calculation can be done when we fix the anchor and the negative instance; thus, we can find $2^{x-y} - 1$ valid triplets. Likewise, $2^{x-y} - 1$ valid triplets contain the fixed positive and negative instances. Now, let us consider the number of forces used in a batch update that is caused by any two instances in the batch. We first consider the case where these two instances belong to the same class. They should attract each other with the force whose magnitude is equal to $\|F_{attract}\|$. The link between these instances is established $2^{x-y+1} \cdot (2^y - 1)$ times in any batch if all the valid triplets are utilized; thus, the force affecting on the first instance is proportional to $2^{x-y+1} \cdot (2^y - 1) \cdot \|F_{attract}\|$ because of the forces denoted by $(f_a - f_p)$, and $(f_p - f_a)$ in the partial derivatives of the loss functions. On the other hand, when these two instances are from different classes, they repel each other with a force whose magnitude is $\|F_{repel}\|$. The triplet loss functions in the literature, utilize the forces due to only the terms $(f_a - f_n)$, and $(f_n - f_a)$ for repelling, whereas there are 4 terms that are being exploited in our proposed loss function for the same purpose, which are $\rho \cdot (f_a - f_n)$, $\rho \cdot (f_n - f_a)$, $\rho \cdot (f_p - f_n)$, and $\rho \cdot (f_n - f_p)$ with their weights. Hence, the repelling force between these instances affecting on the first instance is proportional to $2 \cdot (2^{x-y} - 1) \cdot \|F_{repel}\|$ and $2^2 \cdot (2^{x-y} - 1) \cdot \rho \cdot \|F_{repel}\|$

for the losses in the literature and our proposed loss, respectively.

At this point, we *assume that the ideal condition happens when the effect of positive and negative force is equal to each other*. It means $2 \cdot (2^{x-y} - 1)$ should be equal to $2^{x-y+1} \cdot (2^y - 1)$ when $\|F_{attract}\| = \|F_{repel}\|$ in the losses in the literature. Then $x = \log_2(\frac{2^y}{2-2^y})$ but there is no such $y > 0$ integer value that satisfies this equality. In our loss, the condition $2^{x-y+1} \cdot (2^y - 1) = 2^2 \cdot (2^{x-y} - 1) \cdot \rho$ can be satisfied if ρ is chosen as in Equation (3.19).

$$\rho = 2^{x-y-1} \cdot \frac{2^y - 1}{2^{x-y} - 1}. \quad (3.19)$$

For instance, when the $x = 5$ and $y = 2$, the ideal $\rho = 12/7$, which is the setting we used in the experiments. However, 0.5, 1, and 2 values were also tried in the experiments to show the effect of this hyper-parameter. More detail about these experiments is given in Chapter 4. After choosing the ρ value, we should find the ideal r and s values, that is explained in the next subsection, but before that, the equilateral triangle case for the gradients are plotted for these ρ values in Figure 3.4. The motivation behind plotting the different values of ρ in the equilateral triangle case is that this case eliminates the effect of r and s since all the items in a triplet are affected by each other from the same distance.

3.2.2 Discussion on the Power of the Forces

In this subsection, we discuss the effects of the values of r and s on the region that is determined by the in-loss-miner (due to the hinge loss and the margin). The surface of this region can be found by the boundary conditions which are shown in Equation (3.20).

$$\begin{aligned} \frac{1-r}{(s+1) \cdot \rho} \cdot \|f_a - f_p\|_2^{s+1} + \frac{1-r}{\rho} \cdot m_1 &= \|f_a - f_n\|_2^{1-r} + \|f_p - f_n\|_2^{1-r}, \\ \frac{1-r}{(s+1) \cdot \rho} \cdot \|f_a - f_p\|_2^{s+1} + \frac{1-r}{\rho} \cdot (m_1 - m_2) &= \|f_a - f_n\|_2^{1-r} + \|f_p - f_n\|_2^{1-r}. \end{aligned} \quad (3.20)$$

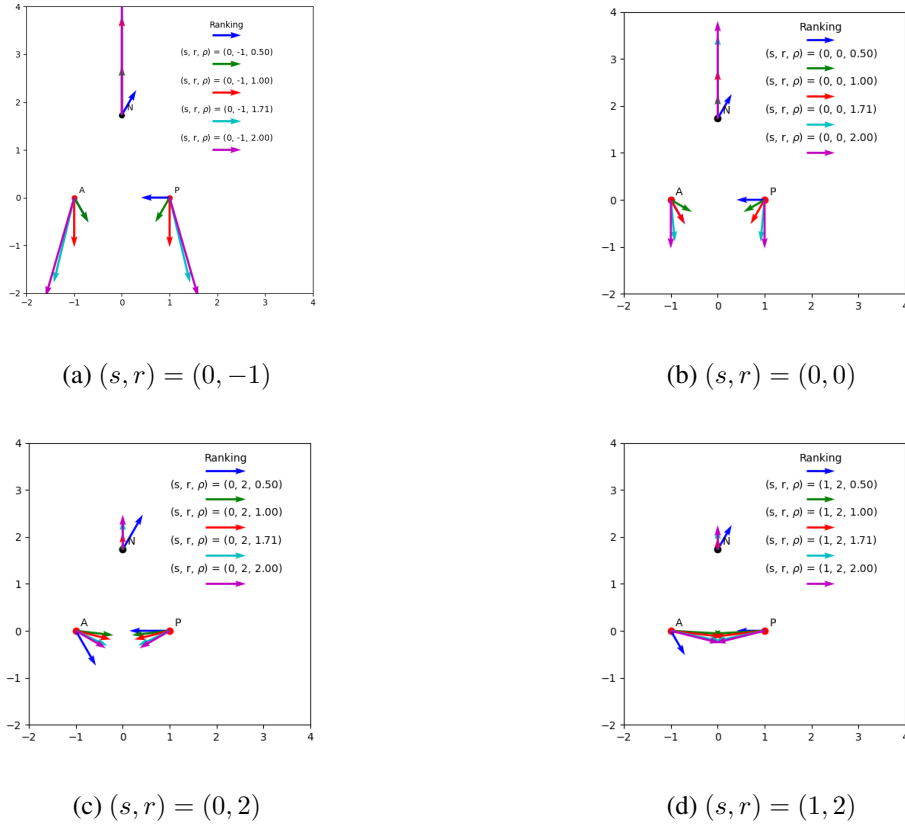
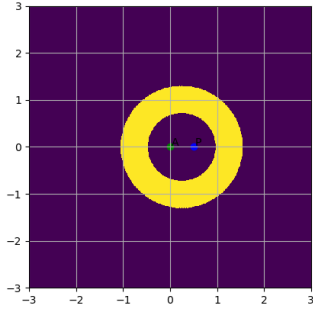
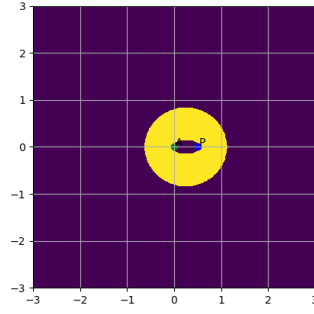


Figure 3.4: The 2-dimensional representation of the embedding space when the anchor instance (denoted as A) is at $(-1, 0)$, the positive one (P) is at $(1, 0)$, and the negative (N) is at $(0, \sqrt{3})$. The sub-figures show the different ρ values.

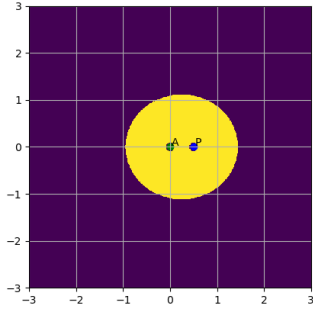
Consider the case of the left hand side of (3.20) is fixed, i.e. the locations of the anchor and the positive instances are fixed, $\|f_a - f_p\|_2 = 0.5$, $s = 0$, and $\rho = 12/7$. Now, we can analyze the effects of the power r on the region of the negative instances that are able to cause non-zero loss value. First, we should find the search space (feasible region) for r . It is a non-negative integer number because of the motivation of our loss function, that is explained in Subsection 3.1.3, and also shown in Equation (3.15). Moreover, $r \neq 1$ from Equation (3.17). Then, we should check the cases, $r = 0$, and $r > 1$ where $r \in \mathbb{Z}^+$. The 2-dimensional plots of the filtered regions by in-loss-miner when $r = -1$, $r = 0$, $r = 2$, and $r = 3$ are shown in Figure 3.5.



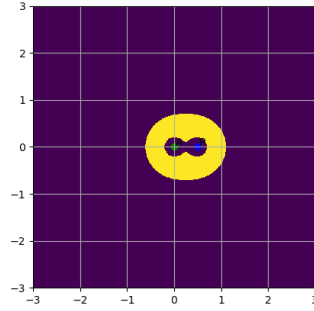
(a) $r = -1, m_1 = 2.5, m_2 = 2.0$



(b) $r = 0, m_1 = 2.5, m_2 = 2.0$



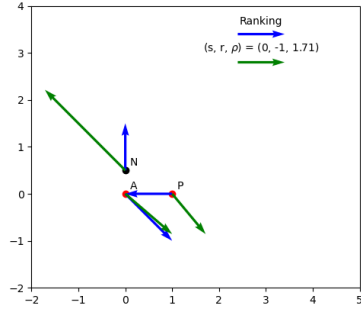
(c) $r = 2, m_1 = -3.5, m_2 = 20.0$



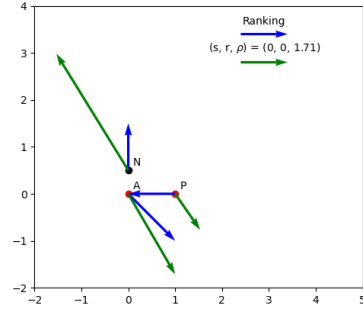
(d) $r = 3, m_1 = -3.5, m_2 = 20.0$

Figure 3.5: The 2-dimensional representation of the mined regions by in-loss-miner for the negative items when the anchor instance (denoted as A) is at $(0, 0)$, the positive one (P) is at $(0.5, 0)$. The sub-figures show the different r values. The other hyper-parameters are as follows: $s = 0, \rho = 12/7$. The negative items in the yellow region are being affected by force caused by both anchor and positive instances.

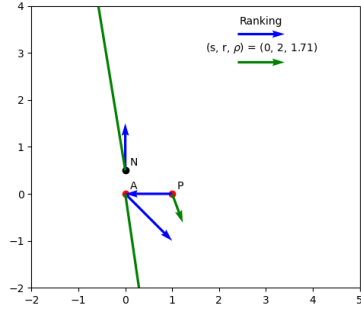
As shown in Figure 3.5, the larger r value causes the smaller area of the filtered region, which means the in-loss-miner filters out almost all the triplets and restrains the training. In order to prevent this situation m_1 and m_2 parameters should be adjusted which are discussed in Section B. Moreover, the gradients should also be considered. The case that the anchor, positive and negative items are at $(0, 0)$, $(1, 0)$, and $(0, 0.5)$ is shown in Figure 3.6.



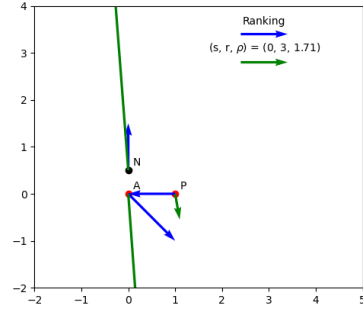
(a) $r = -1$



(b) $r = 0$



(c) $r = 2$



(d) $r = 3$

Figure 3.6: The 2-dimensional representation of the embedding space when the anchor instance (denoted as A) is at $(0,0)$, the positive one (P) is at $(1,0)$, and the negative (N) is at $(0,0.5)$. The sub-figures show the different r values. The other hyper-parameters are as follows: $s = 0$, $\rho = 12/7$.

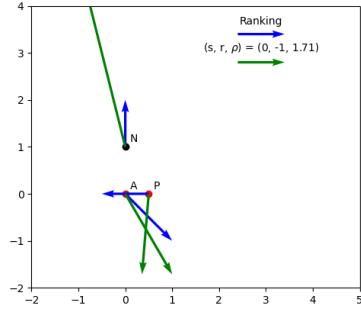
Figure 3.6 shows the effect of r value when the other parameters are fixed. Obviously, the increase of r leads to the rise of magnitude of repelling forces between $f_a \leftrightarrow f_n$ and $f_p \leftrightarrow f_n$. Since the negative item is closer to the anchor than the positive one, the direction of the overall gradient acting on the anchor rotates clockwise when r increases. Another observation is that the direction of the negative item also rotates clockwise since the ratio of forces between $f_a \leftrightarrow f_n$ and $f_p \leftrightarrow f_n$ increases when r rises, which is desired as mentioned in Section 3.1.3.

In contrast, another information utilized while determining the r value is that the study [42] claims that the embeddings which are close to each other have high noise in the gradients. If the r value increases, this noise may be amplified, which makes the model collapse as mentioned in [42]. Although the m_2 parameter prevents these

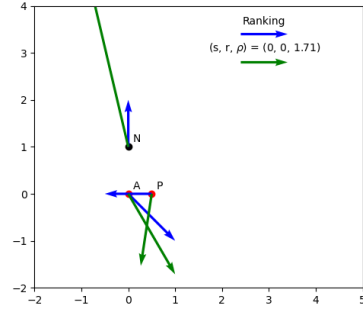
noises, we take into consideration this situation while adjusting r . Therefore, in the experiments, we use $-1, 0, 2$ values for r .

The value of s is also a positive integer due to the same reason. In order to determine its value, we should consider the latter epochs of the training process. The objective of the loss function is to repel the objects in different classes and attract the samples belonging to the same class. Hence, in the latter epochs, the distance of $\|f_a - f_p\|_2$ will get smaller. Let us test two cases, that are $s = k$ and $s = l$ where $k, l \in \mathbb{Z}^{\geq 0}$ and $k < l$. When the condition $\|f_a - f_p\|_2 < 1$ is satisfied (it will be satisfied because of the normalization layer on top of our neural network, explained in Chapter 4), $\|f_a - f_p\|_2^l < \|f_a - f_p\|_2^k$. The smaller value of $\|f_a - f_p\|_2^l$ may cause the smaller region for selecting negatives and eliminates the more number of negative items when comparing with the case of $s = k$.

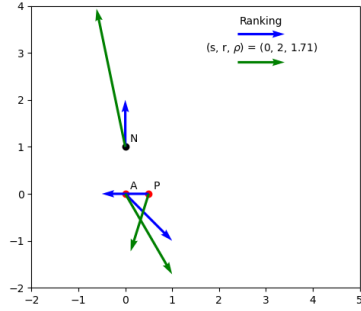
On the other hand, the s value should be large enough for holding the motivation mentioned in Section 3.1.3. This hypothesis is tested by plotting the gradients for various points of negative items in 2-dimensional space shown the case that the anchor, positive and negative items are at $(0, 0)$, $(0.5, 0)$, and $(0, 1)$ is shown in Figure 3.7.



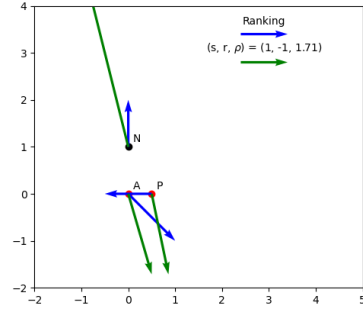
(a) $(s, r) = (0, -1)$



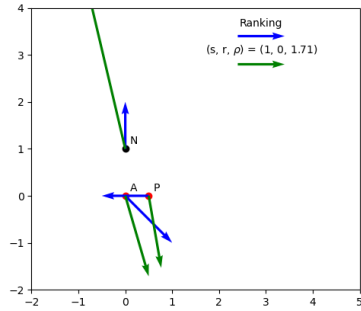
(b) $(s, r) = (0, 0)$



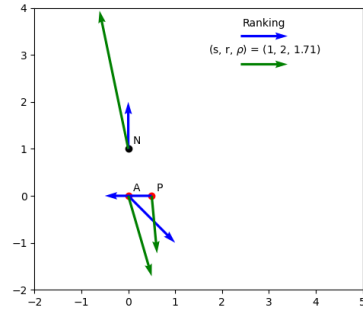
(c) $(s, r) = (0, 2)$



(d) $(s, r) = (1, -1)$



(e) $(s, r) = (1, 0)$



(f) $(s, r) = (1, 2)$

Figure 3.7: The 2-dimensional representation of the embedding space when the anchor instance (denoted as A) is at $(0, 0)$, the positive one (P) is at $(0.5, 0)$, and the negative (N) is at $(0, 1)$. The sub-figures show the different (s, r) values. The hyper-parameter ρ is equal to $12/7$.

All in all, we choose s as 0, 1 and r as $-1, 0, 2$ in the experiments with various ρ values selected in Section 3.2.1. Note that $(s, r, \rho) = (1, -1, 1)$, $(s, r) = (0, 0)$ and $(s, r) = (1, 2)$ are the settings of entangle, modified entangle and metric learning with

entangled triplet unified (METU) loss respectively.

3.2.3 Discussion on the m_1 and m_2 for various (s, r, ρ) 's

In the same vein of Section B, we adjust the m_1 and m_2 parameters empirically. It means we adjust these values for the case where $\|f_a - f_p\|_2 = 0.5$. We take the ranking loss' in-loss-miner as a baseline. After finding the region for ranking loss, the m_1 is determined so that the region for the proposed loss encapsulates this region. However, we have not found an analytically reasonable way to determine the value of m_2 . Actually, we use this parameter to avoid the model from the gradients with high noise. Hence, we train the models with several different m_2 parameters for showing the difference due to its effects. We have not noticed a good value yet.

There are 5 hyper-parameters to determine. We adjust 5 experiments whose parameters are shown in Table 3.4 because of the same reasoning mentioned in B, that is the region of the in-loss-miner includes the region for the ranking loss. The region generated by in-loss-miner with the chosen hyper-parameters is similar to the figure in B.1.

Table 3.4: The table for the hyper-parameters of the experiments we designed.

Experiment #	s	r	ρ	m_1	m_2
1	1	2	1.0	-2.0	5.0
2	1	2	1.0	-2.0	10.0
3	1	2	1.0	-2.25	5.0
4	1	2	1.0	-2.25	10.0
3	1	2	1.0	-2.5	5.0
4	1	2	1.0	-2.5	10.0
5	1	2	1.0	-3.0	5.0
6	1	2	1.0	-3.0	10.0

CHAPTER 4

EXPERIMENTS AND DISCUSSIONS

This chapter is about the experiments that are performed to test our predictions mentioned in Chapter 3. Within the scope of this study, we implement a framework in order to compare the methods mentioned in Chapter 2 with our proposed loss function and the mining methods in Chapter 3. We utilize a dataset that is widely used in this field as discussed in Section 2.3; therefore, there are several tables of the results that are related to this dataset and one more table for the toy dataset generated by Gaussian distribution. Moreover, this dataset is split into two, namely, train-validation and test sets with an open set scenario in order to re-create the conditions of the image retrieval problem. In order to design these experiments in a fair way, the architectures of the models, trained with various methods in the literature, are fixed. Furthermore, the stopping conditions and optimizers are chosen the same in all of the experiments in order to avoid the effects on the performance scores of the methods. Therefore, the improvements are directly caused by the design of the loss function, and the mining methods are shown in the quantitative results of the experiments.

This chapter is divided into three parts. First, the experimental setup and configurations of them are explained in Section 4.1. Next, the results of the experiments are shown in Section 4.2. Finally, the discussions on the results are in Section 4.3.

4.1 Experimental Setup

The experiments are designed in a way such that a fair comparison can be made. Therefore, several properties were fixed during training, such as optimizer, batch size,

etc. A framework was implemented to have full control over the experiments. The implementations of this framework are explained in Appendix A. Also, the testing platform and toy dataset generation procedure are explained in detail below.

4.1.1 Testing Platform

In the testing platform, a tester class was designed which calculates all of the retrieval scores mentioned in Section 2.2. Moreover, the test score of the best model among the 4 fold leave-one-out cross-validation is reported in the tables of the results. The experiments were performed on a commonly used dataset: CUB-200-2011 [40]. Moreover, the methods were tested on a toy example dataset that all of the samples are generated from a Gaussian distribution which will be described in the next subsection.

Furthermore, the configuration file for the experiments is fixed in the experiment objects. In order to compare the effectiveness of the methods, the optimizer was fixed as *RMSprop* with the learning rate as 10^{-6} for both feature extractor and the embedding layer of the network. The batch size is 32 in the training phase. When the online sampler is utilized, the number of iterations is selected as 1000, and the number of classes in a batch is 4 (hence, there are $32/4 = 8$ samples that are in the same class in a batch). The network architecture for the feature extraction is chosen as *ResNet-18* (which is one of the light-weight network architecture) that is pre-trained on the ImageNet dataset [5] due to memory concerns. The embedding layer is a *linear layer* with an input size of 1000 and an output size of 128 (which is equal to the dimension of the embedding space) and is initialized with the method mentioned in the study [12]. The open-set splitter with a test ratio of 0.5 and 4 fold leave-one-out cross-validation was used. In other words, the train, validation, and the test sets include $3n$, n , and $4n$ samples, respectively, when the dataset size is $8n$. The stopping condition is two-fold: The experiment is stopped when the epoch number exceeds the maximum epoch number (chosen as 500). The second condition is the patient epoch. It means when the model has not found the parameters whose validation score exceeds the last best score for 100 epochs, then the experiment is stopped. At that point, there are plenty of scores evaluated on the validation set. We use the *MAP@R* score for the patient epoch condition, like the study [26].

4.1.2 Data Generation and Network Architecture for the Toy Dataset

In order to show the effectiveness of this loss function, we design several experiments with a toy dataset. The items in this dataset are generated with the help of Gaussian distribution. First, the means of each class are determined with the uniform distribution between 0 and 1. After that, the 3-dimensional vectors of each item are sampled from a multivariate normal distribution whose covariance matrix's off-diagonal entries are 0. The diagonal entries of them are equal to $2 \cdot 10^{-2}$. There are 32 classes in total, and each class has 200 samples.

The network architecture consists of two linear layers. There is a leaky ReLU activation function in-between the layers in order to make the model a non-linear transform. The dimension of the hidden layer is 32.

4.2 Experimental Results

In this section, the experimental results of several loss functions are tabulated in the tables. All of these experiments are performed using the settings described in Section 4.1. They are designed to test the hypothesis claimed in Chapter 3 without any external effect such as mining methods, network architecture, etc. Also, we try to indicate the effect of the hyper-parameters of both the losses in the literature and the proposed losses. First, the experiments on the toy dataset are performed in order to show the effectiveness of the idea behind the proposed losses in Section 4.2.1. Then, the experiments for the losses found in the literature are performed with various hyper-parameters, whose results are shown in Section 4.2.2. Finally, in order to analyze the proposed loss function and its hyper-parameters, several experiments were conducted with different values of the hyper-parameters in Section 4.2.3, which is discussed in Section 3.2.

4.2.1 Experiments on the Toy Dataset

The experiments described in Section 4.1.2 are performed. The results are tabulated in Table 4.1.

Table 4.1: The train results of the triplet loss function methods in the literature trained and tested in our framework on the *toy* dataset with various hyper-parameters (denoted as *HP*).

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Scores on the Input Space									
-	70.25	84.42	92.58	96.96	69.50	68.72	67.89	54.76	31.77
Initial Scores on the Feature Space									
-	59.17	74.17	85.21	92.71	58.00	56.89	54.91	35.94	21.07
Ranking Loss [36]									
$m = 0.01$	67.79	82.67	91.25	96.17	67.52	67.01	66.44	53.88	37.32
$m = 0.1$	70.04	83.67	92.75	96.38	69.35	69.35	68.65	56.90	40.75
$m = 2.5$	69.54	83.54	92.67	96.58	68.62	67.39	66.28	48.67	32.00
$m = 40.0$	69.54	83.33	92.71	96.46	68.52	67.33	66.18	48.30	31.62
Distance Sensitive Loss with $(s, r, \rho, m_1) = (1, 2, 1.0, -2.25)$									
$m_2 = 5.0$	70.21	84.62	92.54	96.79	69.44	68.82	68.00	55.35	38.88
$m_2 = 10.0$	70.29	84.58	92.58	96.67	69.44	68.91	67.95	55.36	38.90
Distance Sensitive Loss with $(s, r, \rho, m_1) = (1, 2, 1.0, -2.5)$									
$m_2 = 5.0$	70.08	84.42	92.46	96.88	69.42	68.83	68.03	55.28	38.81
$m_2 = 10.0$	70.17	84.38	92.46	96.79	69.35	68.91	67.97	55.28	38.81
Distance Sensitive Loss with $(s, r, \rho, m_1) = (1, 2, 1.0, -2.75)$									
$m_2 = 5.0$	69.96	84.58	92.46	96.79	69.50	68.76	67.98	55.22	38.74
$m_2 = 10.0$	69.79	84.50	92.46	96.75	69.42	68.78	67.95	55.23	38.76

The results in Table 4.1 show the idea behind the design of the distance sensitive loss is quite consistent. Both the train, validation and test scores surpass the scores of the vanilla loss function (ranking loss). However, only the train results are meaningful since the experiments are performed with the open-set scenario. Hence, the results in Table 4.1 show only the scores on train set. In addition, the ranking loss function does not reach the level of the input space at the end of the training, whereas our proposed loss achieves with certain hyper-parameters.

4.2.2 Experiments for the Loss Functions in the Literature

The experiments on the CUB-200-2011 dataset [40] for the prominent triplet based losses from the literature are shown in Tables 4.2 and 4.3. The experiments are executed with various hyper-parameters, in which the parameter set includes the best values mentioned in the studies of the methods.

Table 4.2: The test results of the triplet loss function methods in the literature trained and tested in our framework on *CUB200-2011* dataset [40] with various hyper-parameters (denoted as *HP*). (Part-1)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Original Triplet [14]									
-	35.79	47.89	60.40	71.84	33.50	30.78	27.27	16.32	7.98
Ranking [36]									
$m = -0.2$	33.15	45.71	59.05	72.60	30.78	27.94	25.12	15.07	6.50
$m = -0.1$	36.46	49.61	62.32	75.42	34.25	31.32	28.33	17.25	8.00
$m = 0.0$	50.71	65.02	77.28	86.16	48.72	45.76	41.75	25.77	14.77
$m = 0.01$	52.80	65.61	77.04	85.96	50.51	47.61	43.86	27.73	16.65
$m = 0.1$	50.49	63.12	74.48	84.18	48.22	45.27	42.00	26.84	16.03
$m = 0.2$	47.45	60.33	72.06	82.07	45.54	42.76	39.75	25.70	15.09
$m = 0.3$	43.18	56.28	68.38	79.14	41.46	39.23	36.13	23.77	13.46
$m = 2.5$	34.45	46.74	59.32	70.91	32.20	28.84	25.29	14.71	6.69
$m = 40.0$	34.69	46.62	58.91	70.80	32.20	28.77	25.12	14.69	6.70
FaceNet [28]									
$m = 0.01$	49.97	63.13	75.44	84.98	47.96	45.58	42.38	26.96	16.04
$m = 0.1$	50.73	64.23	74.76	84.42	48.81	45.65	42.41	27.30	16.36
$m = 0.2$	49.17	62.61	74.12	84.23	47.36	44.78	41.71	26.62	15.83
$m = 0.3$	49.61	62.00	74.00	83.42	47.35	44.61	41.22	26.41	15.70
$m = 2.5$	35.65	47.65	59.59	70.88	33.25	30.19	26.81	16.18	7.84
$m = 40.0$	34.54	46.29	59.20	71.44	32.10	28.92	25.73	14.84	6.84
N-Pairs [31]									
-	36.56	48.36	60.99	72.65	34.21	31.46	28.02	16.83	8.34

As seen in Table 4.2, some of the well-known loss functions have not any hyper-parameter to optimize such as Original Triplet [14], and N-Pairs [31]. Hence, they are trained with a single experiment. On the other hand, several experiments are performed with these losses and various values of their hyper-parameters. The results of these experiments show that the ranking [36], and FaceNet [28] losses are quite sensitive to the variations of the hyper-parameters. Moreover, the experiments with several negative and positive values of the m , are also performed in order to check if the best hyper-parameters given in the previous studies are valid or not. As expected, we got results that are consistent with these studies.

Table 4.3: The table for the test results of the triplet loss function methods in the literature trained and tested in our framework on *CUB200-2011* dataset [40] with various hyper-parameters (denoted as *HP*). (Part-2)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Ratio [41]									
$m = 0.01$	52.31	65.67	76.37	85.28	50.42	47.41	43.67	27.33	16.53
$m = 0.1$	49.29	62.34	74.05	84.00	47.57	45.10	41.87	26.63	15.89
$m = 0.2$	47.25	60.01	72.10	82.01	45.31	42.74	39.81	25.56	14.97
$m = 0.3$	43.55	57.44	69.11	79.15	42.71	40.02	36.86	23.97	13.74
$m = 40.0$	39.21	51.00	62.53	73.94	37.16	34.32	31.15	19.68	10.52
Angular [37]									
$\rho = 0.2$	27.55	39.08	52.40	65.55	25.08	22.33	19.72	11.77	4.43
$\rho = 0.3$	30.11	42.20	55.72	69.14	28.06	25.41	22.58	13.79	5.60
$\rho = 0.4$	40.04	52.68	65.60	77.63	37.48	35.01	31.85	19.66	9.84
$\rho = 0.5$	45.66	58.64	72.18	82.53	43.40	40.87	37.40	23.07	12.6
$\rho = 0.6$	46.56	60.65	73.43	83.34	44.71	41.62	37.96	23.19	12.70
$\rho = 0.63$	41.75	54.90	68.16	79.46	39.15	36.20	32.75	19.81	10.05
$\rho = 0.8$	28.49	40.78	53.78	67.17	26.51	23.89	21.34	12.68	5.07
$\rho = 0.96$	26.38	38.12	51.67	65.21	24.37	22.28	19.62	11.44	4.36
$\rho = 2.0$	26.52	38.44	51.86	65.41	24.61	22.51	19.77	11.47	4.35

Moreover, we conduct the experiment with angular loss function without hinge loss shown in Equation (2.12). The results are shown in Table 4.4.

Table 4.4: The table for the test results of the angular triplet loss function without hinge function trained and tested in our framework on *CUB200-2011* dataset [40] with various hyper-parameters (denoted as *HP*).

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Angular without hinge function in (2.12)									
$\rho = 0.2$	27.36	38.99	52.16	65.51	25.16	22.48	19.96	11.84	4.54
$\rho = 0.3$	30.05	42.45	55.98	69.04	28.12	25.67	22.32	12.11	4.85
$\rho = 0.4$	33.19	45.27	57.95	70.16	30.49	27.63	24.34	14.05	6.20
$\rho = 0.5$	33.69	45.97	58.29	70.04	31.50	28.45	25.16	14.52	6.64
$\rho = 0.6$	34.67	47.35	59.42	71.37	32.59	29.35	26.07	15.12	7.03
$\rho = 0.63$	34.49	45.91	58.73	70.73	31.71	28.85	25.49	14.76	6.78
$\rho = 0.79$	34.08	45.66	58.34	70.73	31.68	28.89	25.68	14.86	6.88
$\rho = 0.8$	33.81	46.15	58.69	70.48	31.61	28.73	25.70	14.71	6.80
$\rho = 0.96$	34.47	46.29	58.14	70.07	32.00	28.87	25.51	14.93	6.95

The difference between the scores of the angular loss in Tables 4.3 and 4.4 denotes the advantage of usage of hinge function. It means the in-loss-mining process progresses the model since it causes eliminating the redundant triplets while calculating the loss value.

In addition, the experiments for one of the state-of-the-art methods in deep metric learning named as moving loss, in Equation (2.14), are also performed, whose results are shown in Tables 4.5, 4.6 and 4.7.

Table 4.5: The table for the test results of the moving triplet loss function methods in the literature trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*). (Part-1)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Moving with $\rho = 0.1$									
$m = 0.01$	35.74	48.75	61.95	74.92	33.38	30.77	27.94	17.04	7.88
$m = 0.1$	50.20	63.50	75.07	84.91	48.48	45.43	42.32	27.03	16.13
$m = 0.2$	49.38	62.44	74.38	84.54	47.54	45.25	42.14	26.89	15.99
$m = 0.3$	49.32	62.34	74.02	83.88	47.30	44.89	41.81	26.68	15.91
$m = 2.5$	35.26	47.57	59.42	70.48	33.04	30.00	26.57	16.28	7.89
$m = 5.0$	34.45	46.46	59.27	70.73	31.82	28.73	25.54	14.74	6.77
$m = 10.0$	34.22	45.85	58.25	69.77	31.36	28.38	25.00	14.61	6.70
$m = 20.0$	33.73	45.63	58.44	70.14	31.59	28.51	25.05	14.48	6.61
$m = 40.0$	33.91	45.93	58.34	70.54	31.74	28.40	25.09	14.55	6.61
Moving with $\rho = 0.2$									
$m = 0.01$	33.58	46.25	60.04	72.81	31.33	28.84	25.88	15.74	6.97
$m = 0.1$	45.17	59.44	72.69	83.71	42.94	40.06	36.57	22.30	11.66
$m = 0.2$	50.35	63.67	75.00	84.64	48.55	45.73	42.31	27.16	16.21
$m = 0.3$	49.56	62.74	74.97	84.62	47.90	45.45	42.05	26.86	16.08
$m = 2.5$	36.34	48.28	60.53	71.66	33.85	30.75	27.45	16.48	8.07
$m = 5.0$	34.49	46.61	58.56	70.32	32.03	28.60	25.14	14.69	6.72
$m = 10.0$	34.42	45.75	58.09	70.12	31.57	28.65	25.20	14.62	6.73
$m = 20.0$	34.28	45.95	58.58	71.00	31.47	28.53	25.35	14.66	6.68
$m = 40.0$	33.64	45.29	58.47	69.99	31.16	28.37	25.16	14.60	6.69

Table 4.6: The table for the test results of the moving triplet loss function methods in the literature trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*). (Part-2)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Moving with $\rho = 0.3$									
$m = 0.01$	32.68	45.31	58.47	71.57	30.57	27.68	24.72	15.06	6.52
$m = 0.1$	34.91	47.54	60.90	73.77	32.68	29.87	27.04	16.60	7.56
$m = 0.2$	50.52	63.98	74.97	84.66	48.93	46.19	42.41	27.21	16.32
$m = 0.3$	49.66	62.76	74.54	84.59	47.99	45.19	42.20	26.93	16.06
$m = 2.5$	34.77	46.32	59.13	70.88	32.53	30.02	26.66	16.36	7.98
$m = 5.0$	33.88	45.86	58.17	70.36	31.36	28.27	25.06	14.31	6.45
$m = 10.0$	33.44	45.90	57.87	69.21	31.09	28.05	24.73	14.23	6.46
$m = 20.0$	33.61	45.80	58.36	70.32	31.21	28.37	25.13	14.43	6.56
$m = 40.0$	34.49	46.62	59.33	71.84	31.9	29.07	25.78	14.89	6.83
Moving with $\rho = 0.4$									
$m = 0.01$	31.26	44.02	57.38	71.51	29.33	26.81	24.00	14.37	6.03
$m = 0.1$	33.12	45.88	58.91	71.89	30.82	27.82	25.07	15.18	6.59
$m = 0.2$	45.09	59.27	72.27	83.58	43.10	40.19	36.50	22.39	11.74
$m = 0.3$	50.51	63.05	74.92	84.52	48.13	45.46	42.40	27.09	16.20
$m = 2.5$	35.13	47.10	59.15	71.22	33.04	30.05	26.77	16.23	7.88
$m = 5.0$	34.01	45.93	58.46	70.63	31.50	28.25	25.26	14.50	6.58
$m = 10.0$	34.64	46.42	58.22	70.58	31.56	28.40	25.35	14.41	6.57
$m = 20.0$	34.23	46.34	58.78	70.63	31.83	28.88	25.35	14.53	6.65
$m = 40.0$	34.33	45.93	58.59	70.48	31.52	28.47	25.19	14.46	6.55

Table 4.7: The table for the test results of the moving triplet loss function methods in the literature trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*). (Part-3)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Moving with $\rho = 0.5$									
$m = 0.01$	30.91	43.35	56.74	70.12	28.60	25.97	23.15	13.89	5.77
$m = 0.1$	32.12	44.43	58.25	71.03	29.51	27.17	24.16	14.62	6.18
$m = 0.2$	34.81	47.50	60.50	73.63	32.26	29.45	26.60	16.14	7.23
$m = 0.3$	50.88	64.06	75.56	84.62	48.78	45.83	42.41	27.13	16.22
$m = 2.5$	35.42	47.23	59.37	70.21	33.39	30.38	26.81	16.27	7.96
$m = 5.0$	33.17	45.26	57.83	69.65	30.76	27.80	24.53	14.09	6.30
$m = 10.0$	33.98	45.81	58.74	70.00	31.48	28.37	25.04	14.30	6.49
$m = 20.0$	33.68	45.80	57.56	69.77	31.39	28.17	24.90	14.01	6.36
$m = 40.0$	34.37	46.37	58.95	71.08	31.74	28.71	25.43	14.48	6.61

The results of the moving loss are also consistent with the paper [3]. The gap between the score values may be due to the difference in the architecture of the bone network used in the experiments.

4.2.3 Experiments for the Proposed Loss Function

Several experiments were designed for analyzing the distance sensitive entangled triplet loss function in (3.18). It has 5 hyper-parameters, which can be optimized with grid search, random search, or Bayesian optimization. However, in order to find the logical reasons, we rigorously focus on a different parameter in each experiment. At that point, we assume a presumption that the dependence between the hyper-parameters is weak due to the evidence of the empirical observations. First, we try to eliminate the effect of the margins m_1 , m_2 , and the hinge function. Hence, the experiments in Table 4.8 are performed by selecting $m_1 = 40$ and $m_2 = 1000$, which leads to a case that the in-loss-miner does not filter out any triplet.

Table 4.8: The table for the test results of our distance sensitive triplet loss function methods without the effect of the hinge function (which means we selected $m_1 = 40$ and $m_2 = 1000$) trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*).

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Distance Sensitive Loss with $(s, r) = (1, -1)$ and without hinge function									
$\rho = 0.5$	33.90	45.68	58.32	70.46	31.17	28.34	25.18	14.60	6.62
$\rho = 1.0$	34.94	47.05	59.12	70.73	32.29	29.11	25.70	14.86	6.88
$\rho = 12/7$	34.37	46.35	58.96	71.13	31.91	28.86	25.53	14.85	6.82
$\rho = 2.0$	34.01	45.93	58.37	70.80	31.67	28.64	25.47	14.77	6.77
Distance Sensitive Loss with $(s, r) = (0, 0)$ and without hinge function									
$\rho = 0.5$	34.71	46.32	58.91	70.61	31.64	28.59	25.06	14.72	6.67
$\rho = 1.0$	33.79	45.95	58.37	70.44	31.39	28.31	24.91	14.41	6.50
$\rho = 12/7$	34.40	46.20	58.44	70.90	31.83	28.63	25.36	14.56	6.60
$\rho = 2.0$	34.22	46.30	58.86	70.80	31.77	28.75	25.38	14.67	6.64
Distance Sensitive Loss with $(s, r) = (1, 0)$ and without hinge function									
$\rho = 0.5$	34.59	46.69	59.98	71.64	32.22	29.57	26.35	15.47	7.27
$\rho = 1.0$	34.89	46.67	59.82	71.61	32.33	29.57	26.36	15.45	7.29
$\rho = 12/7$	35.62	47.82	60.47	71.76	33.26	30.30	26.68	15.73	7.43
$\rho = 2.0$	35.21	47.18	59.88	71.81	32.67	29.59	26.36	15.62	7.33
Distance Sensitive Loss with $(s, r) = (0, 2)$ and without hinge function									
$\rho = 0.5$	36.85	48.80	60.96	72.77	34.44	31.66	28.17	16.73	8.17
$\rho = 1.0$	37.37	49.11	61.28	73.68	34.86	31.77	28.66	17.07	8.38
$\rho = 12/7$	36.85	49.07	61.29	73.08	34.41	31.62	28.20	16.89	8.31
$\rho = 2.0$	36.24	48.68	61.04	72.20	34.28	31.33	27.76	16.69	8.22
Distance Sensitive Loss with $(s, r) = (1, 2)$ and without hinge function									
$\rho = 0.5$	37.59	49.39	61.87	72.86	35.29	32.30	29.03	18.02	9.21
$\rho = 1.0$	38.34	49.63	62.85	73.84	35.55	32.71	29.61	18.10	9.24
$\rho = 12/7$	37.46	49.21	61.93	72.70	35.07	32.43	29.13	17.91	9.13
$\rho = 2.0$	38.07	49.59	62.20	73.13	35.52	32.74	29.17	18.06	9.26

Table 4.8 shows that the variation in the ρ value does not effect the scores significantly, in contrast to our expectations. It may be due to the fact that there are just a few experiments with changed ρ values, which is because of the limited time budget. On the other hand, the r value is quite a crucial parameter for optimizing the model, and the s value also improves the model when its value increases.

In addition, the scores of the location-aware loss can be seen in Table 4.9. The performance of this loss cannot reach the performance of the distance sensitive loss function. However, it gives comparable results with the state-of-the-art losses in the case without the in-loss-miner.

Table 4.9: The table for the test results of the location-aware triplet loss function methods in the literature trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*).

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Location Aware									
$m = 0.01$	35.25	47.11	59.40	70.95	32.76	29.31	26.22	15.36	7.24
$m = 0.1$	34.39	46.46	58.85	70.48	32.10	29.05	25.81	15.11	7.10
$m = 0.2$	34.91	46.76	59.15	70.66	32.38	29.46	25.96	15.13	7.13
$m = 0.3$	34.10	46.08	58.37	70.88	31.68	28.92	25.75	15.01	6.98
$m = 2.5$	34.77	46.56	59.37	70.51	31.87	29.11	25.49	14.67	6.75
$m = 40.0$	34.44	46.10	58.74	70.49	31.85	29.06	25.82	14.85	6.90

It proves the hypothesis that the effectiveness of the metric learning methods strongly depends on how to use the distance between the instances in a tuple (or batch) in the gradients of the loss function, which is the core hypothesis of this study.

Moreover, the results of distance sensitive loss with various values of m_1 and m_2 are shown in Tables 4.10, 4.11 and 4.12. The experiments are performed with the set of variables are given in Section 3.2.3. At that point, fixing $(s, r) = (1, 2)$ while performing the experiments added the effect of the hinge function. First, we set the $m_1 = -2.5$ which is proposed with the logical reasons mentioned in Section 3.2. The conducted experiments can be seen in Table 4.10.

Table 4.10: The table for the test results of our distance sensitive triplet loss function methods with hinge loss and $m_1 = -2.5$, trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*).

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Distance with $(s, r, \rho) = (1, 2, 0.5)$									
$m_2 = 5.0$	49.12	62.47	75.27	84.71	47.17	44.39	40.64	25.17	14.51
$m_2 = 10.0$	49.78	63.35	75.30	84.82	48.05	45.09	41.13	25.44	14.73
Distance with $(s, r, \rho) = (1, 2, 1.0)$									
$m_2 = 5.0$	49.76	63.30	74.95	84.39	47.84	44.71	40.67	25.17	14.49
$m_2 = 10.0$	49.76	63.28	75.51	84.42	47.48	44.59	40.65	25.11	14.42
Distance with $(s, r, \rho) = (1, 2, 12/7)$									
$m_2 = 5.0$	49.43	62.54	74.83	84.76	47.21	44.53	40.71	25.13	14.43
$m_2 = 10.0$	49.58	63.45	75.49	84.77	47.80	44.77	40.98	25.47	14.73
Distance with $(s, r, \rho) = (1, 2, 2.0)$									
$m_2 = 5.0$	48.77	62.69	74.46	84.82	46.89	44.11	40.69	25.24	14.52
$m_2 = 10.0$	49.70	62.95	74.95	84.32	47.38	44.57	40.80	25.20	14.48

Table 4.11: The table for the test results of our distance sensitive triplet loss function methods with hinge loss, trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*). (Part-1)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.0)$									
$m_2 = 5.0$	44.33	57.70	69.58	79.37	42.97	40.30	37.25	24.36	13.91
$m_2 = 10.0$	44.13	57.17	68.99	79.74	42.66	40.15	37.33	24.42	13.98
$m_2 = 15.0$	45.00	57.16	69.41	79.56	42.83	40.18	37.28	24.35	13.97
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.25)$									
$m_2 = 5.0$	50.41	62.93	74.97	84.44	48.36	45.62	42.41	26.83	16.04
$m_2 = 10.0$	50.25	62.95	74.95	84.01	48.26	45.74	42.21	27.02	16.24
$m_2 = 15.0$	50.14	63.20	74.61	84.00	48.42	45.56	42.41	26.98	16.20
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.5)$									
$m_2 = 5.0$	49.78	62.78	74.48	85.13	47.31	44.57	41.14	25.41	14.65
$m_2 = 10.0$	49.38	62.88	74.54	84.12	47.19	44.35	40.44	25.08	14.40
$m_2 = 15.0$	50.03	63.89	75.89	84.96	47.96	45.12	41.01	25.24	14.54
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -3.0)$									
$m_2 = 5.0$	28.53	40.40	54.44	67.25	26.36	24.08	21.29	12.61	5.01
$m_2 = 10.0$	28.41	40.21	52.73	66.17	26.22	23.45	20.75	12.39	4.87
$m_2 = 15.0$	27.72	39.18	52.82	66.26	25.41	23.35	20.62	12.24	4.79
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -3.5)$									
$m_2 = 5.0$	26.42	38.12	51.55	65.43	24.49	22.15	19.67	11.42	4.34
$m_2 = 10.0$	26.10	38.12	51.74	65.09	24.41	22.31	19.59	11.35	4.29
$m_2 = 15.0$	26.45	38.66	51.47	64.84	24.56	22.19	19.41	11.23	4.25

Table 4.12: The table for the test results of our distance sensitive triplet loss function methods with hinge loss, trained and tested in our framework on *CUB200-2011* dataset with various hyper-parameters (denoted as *HP*). (Part-2)

HP	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.30)$									
$m_2 = 5.0$	50.56	63.50	75.59	85.20	48.73	46.40	43.01	27.22	16.41
$m_2 = 10.0$	50.37	63.50	75.24	84.50	48.60	46.23	42.98	27.24	16.40
$m_2 = 15.0$	50.84	64.01	75.74	84.66	49.21	46.48	43.19	27.17	16.42
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.325)$									
$m_2 = 5.0$	51.76	65.02	76.45	85.40	49.94	47.16	43.82	27.66	16.75
$m_2 = 10.0$	51.08	64.16	76.15	85.30	49.25	46.53	43.32	27.13	16.34
$m_2 = 15.0$	51.67	64.30	75.88	85.08	49.51	46.94	43.61	27.53	16.66
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.35)$									
$m_2 = 5.0$	51.37	63.69	75.79	85.20	49.00	46.74	43.31	27.19	16.34
$m_2 = 10.0$	51.60	64.50	76.32	85.60	49.36	46.81	43.39	27.24	16.39
$m_2 = 15.0$	51.42	64.79	76.55	85.55	49.70	47.16	43.53	27.27	16.43
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.375)$									
$m_2 = 5.0$	51.70	64.62	76.23	85.47	49.64	47.10	43.42	27.18	16.35
$m_2 = 10.0$	51.86	64.38	76.20	85.84	49.59	47.11	43.53	27.19	16.34
$m_2 = 15.0$	51.74	64.47	76.18	85.55	49.56	47.08	43.54	27.30	16.41
Distance with $(s, r, \rho, m_1) = (1, 2, 1, -2.40)$									
$m_2 = 5.0$	51.50	64.96	76.25	85.80	49.59	46.78	43.30	26.97	16.09
$m_2 = 10.0$	51.84	64.92	76.40	85.75	49.69	46.86	43.06	26.98	16.08
$m_2 = 15.0$	51.50	65.16	76.74	85.65	49.86	47.05	43.17	26.98	16.13

4.3 Discussions on Experimental Results

In light of the experimental findings above, we test our hypotheses in two stages. The first stage determines whether the directions and the magnitudes of the gradients we designed to enhance the performance or not. Therefore, we aim to eliminate the effect of in-loss-miners in all the losses in the literature and the proposed losses to

compare. The experiments with $m = 40$ yield the desired scenario; hence, the results of several methods with this configuration are discussed in Section 4.3.1. After this first stage, the best models trained with several loss functions are compared with each other and with the losses proposed in Chapter 3. The discussion about these scores are in Section 4.3.2.

4.3.1 The Effects of the Losses without any Mining

The results from the methods in the literature are taken from the tables in Section 4.2.2. We tabulate the results with the hyper-parameter $m = 40$. However, some loss functions have more than one hyper-parameters, such as moving loss. Then, we use the best score among the test results in Tables 4.5, 4.6, and 4.7. Another issue to mention is that a new m parameter should be added in the ratio and angular losses in order to eliminate the advantage gained from the hinge function. Hence, the scores in Table 4.4 are utilized for the comparison of the gradients without the in-loss-miner. As it can be observed from Table 4.13, the distance sensitive loss with $(s, r) = (1, 2)$ significantly outperforms all the losses proposed in the literature. These observations prove our hypothesis mentioned in Section 3.1.

Table 4.13: The table for the test results of all the triplet loss functions without hinge loss trained and tested in our framework on *CUB200-2011* dataset.

Methods	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Losses in the Literature with $m = 40$									
Original	35.79	47.89	60.40	71.84	33.50	30.78	27.27	16.32	7.98
Ranking	34.69	46.62	58.91	70.80	32.20	28.77	25.12	14.69	6.70
FaceNet	34.54	46.29	59.20	71.44	32.10	28.92	25.73	14.84	6.84
Ratio	39.21	51.00	62.53	73.94	37.16	34.32	31.15	19.68	10.52
N-Pairs	36.56	48.36	60.99	72.65	34.21	31.46	28.02	16.83	8.34
Angular	34.67	47.35	59.42	71.37	32.59	29.35	26.07	15.12	7.03
Moving	34.49	46.62	59.33	71.84	31.9	29.07	25.78	14.89	6.83
Proposed Losses without Hinge Function									
Distance	38.34	49.63	62.85	73.84	35.55	32.71	29.61	18.10	9.24
Location	34.44	46.10	58.74	70.49	31.85	29.06	25.82	14.85	6.90
Entangle	34.94	47.05	59.12	70.73	32.29	29.11	25.70	14.86	6.88
Modified	34.71	46.32	58.91	70.61	31.64	28.59	25.06	14.72	6.67

The first notable finding is about the relation between ranking loss and the distance sensitive loss with $(s, r, \rho) = (1, -1, 1)$. These two cases create similar gradients, which leads to similar performance scores. It causes by the fact that when the in-loss-miner’s effect is ignored (all the valid triplets are utilized in a batch) for calculation of the loss function. In other words, these experiments do not show the effect of utilizing the link between $f_p \leftrightarrow f_n$ in the partial derivative of the loss function. In other words, if a valid triplet (x, y, z) exists with as anchor, positive and negative instances, respectively, then there is also a valid triplet used in the same batch update that is selected as (y, x, z) . Therefore, the ranking and proposed loss with the certain hyper-parameters generate gradients whose directions are the same, and the magnitude of the ranking loss is equal to half of the magnitude of the proposed loss. This difference resembles the change in the learning rate. Indeed, one can conclude that the learning rate does not affect the final performance score of the model.

All the experiments discussed in this subsection suffer from the noisy gradients of the

near instances of the triplets mentioned in [42]. Indeed, the distance sensitive loss function amplifies the noises from the hard-negative samples because of the terms powered by r in (3.17). In spite of its disadvantages, the performance of the distance sensitive loss surpasses the others with a significant gap. This performance might be due to the fact that most of the negatives are located at the distant position to the anchor and positive instances, especially in the later epochs.

In summary, all of the score values are as good as the scores of the losses with the in-loss-miner. When there is no mining, the gradients act on all the positive and negative items wherever they are located in the embedding space, which is 128-dimensional ℓ_2 norm space in our framework. Then, it creates fluctuation, which limits the performance scores of the model. In addition, the normalization of the embeddings may also enhance this effect because it maps all the embeddings into a hyper-sphere or simply maps into a 127-dimensional space. These empirical observations may show the limits that are governed by a theoretical upper-bound for the performance score of a certain loss function. However, finding this limit is out of the scope of this thesis. It is mentioned in Section 5.3 as a future direction of the research.

4.3.2 Comparing Best Results for the Methods

Table 4.14: The table for the best test results of all the triplet loss functions trained and tested in our framework on *CUB200-2011* dataset.

Methods	R@1	R@2	R@4	R@8	P@2	P@4	P@8	P@R	MAP@R
Best Scores for Losses in the Literature									
Original	35.79	47.89	60.40	71.84	33.50	30.78	27.27	16.32	7.98
Ranking	52.80	65.61	77.04	85.96	50.51	47.61	43.86	27.73	16.65
FaceNet	50.73	64.23	74.76	84.42	48.81	45.65	42.41	27.30	16.36
Ratio	52.31	65.67	76.37	85.28	50.42	47.41	43.67	27.33	16.53
N-Pairs	36.56	48.36	60.99	72.65	34.21	31.46	28.02	16.83	8.34
Angular	46.56	60.65	73.43	83.34	44.71	41.62	37.96	23.19	12.70
Moving	50.52	63.98	74.97	84.66	48.93	46.19	42.41	27.21	16.32
Best Scores for Proposed Losses without Hinge Function									
Distance	51.86	64.38	76.20	85.84	49.59	47.11	43.53	27.19	16.34

As it can be observed from Table 4.14, the ranking loss reaches the best performance score among the examined losses from the literature. Although the methods in this table are trained and tested in their networks designed for favoring themselves (as mentioned in [26]), the results in this table are obtained by our framework. Similar to the study in [26], all the valid triplets generated from the batch samples are utilized in the training phase. Therefore, the scores may be different from the values announced in the original papers of the method. Nevertheless, since all of the effects other than loss function are fixed, such as network architecture, the results can be fairly comparable with the other scores in the table.

Table 4.14 shows that the distance sensitive loss function has some promising results, which may surpass the scores of the ranking loss with some certain hyper-parameters. However, the scores of the ranking loss outperform all the others with a roughly 2% gap in *Recall@1*, except the ratio loss. It can be concluded that the proposed loss functions are quite fragile to the variation of the hyper-parameters. Hence, the entire feasible region for this hyper-parameter space should be searched to find the best set.

CHAPTER 5

CONCLUSIONS

5.1 Summary

In this study, the limitations in deep metric learning are investigated deeply by focusing on the gradients of the loss functions as a starting point. An analogy between the electrostatic forces acting on the point charged particles in free space and these gradients of the loss function is established for analyzing the cases in the main problem of deep metric learning. From the classical mechanics' point of view, all the particles in any physical system (does not matter if it is a charged body or a sub-atomic particle) move with the forces corresponding to the principle of minimum action. Roughly speaking, the vector of the force acting on a certain particle is able to be calculated by finding the gradients of the potential energy with respect to its position. The other domain, deep metric learning, is quite similar if the concepts in physics named as the location of a particle, force, potential energy, and movement are converted to the embedding vector, partial gradients of the loss function, loss value, and update, respectively. Hence, one may assume that designing the loss function by considering this analogy can enhance the performance of a model that is trained with the minimum number of updates, which prevents the model from stacking in a local minimum or a saddle point.

To elaborate, the embedding space in deep metric learning is transferred to a medium as if its permittivity constant is negative for zero frequency. In this medium, the electrostatic forces hold the charged particles with the same sign together, whereas they push the charged particles with the reverse sign. We notice it resembles the

main objective of the deep metric learning and build a hypothesis that a new loss designed with the help of this analogy may improve the performance. Hence, three novel loss functions are proposed: Entangle, location-aware, and distance sensitive triplet losses.

5.2 Conclusions

The experiments on three datasets are performed in order to test our hypothesis and show the effectiveness of the methods in the literature and one of the proposed losses. The models are trained and tested by the framework that is implemented by us to fully control the experiments in a fair manner. All the methods are examined within the fixed environment in order to compare the effect of the loss function solely. The results do not reject our aforementioned hypothesis (the alternative hypothesis) if all the losses are utilized without in-loss-miner. Moreover, the proposed losses with the in-loss-miner reach competitive results with the certain hyper-parameters. This promising study shows that the new loss functions are fragile against the variation of the hyper-parameters.

In a nutshell, our conclusions are 2-fold. The first one is as follows: In-loss-miner is quite effective on the performance scores. It means selecting the instances in the tuples in a smart way prevents the fluctuations of the location of the samples in the embedding space due to the unnecessary updates. Hence, it may enhance the performance of the system.

The second conclusion indicates quite a close relationship between online mining techniques and the loss function designs. Indeed, any online mining method can be assembled inside the loss function with a rigorous design of the partial gradients. However, there is no way of modifying the loss function for adding an offline miner. Therefore, designing the offline miner for constructing the batches is still an open question.

5.3 Future Work

Further studies may be about designing an offline miner (which corresponds to the batch sampler in the PyTorch platform) in this framework. Moreover, the effects of these mining methods on these loss functions could be analyzed. Additionally, the fully entangled form, such as the study [31], can be designed in order to improve the effectiveness of the proposed loss functions. These loss functions might be able to utilize the information in all of the batches and find a smarter update for optimization. Finally, the hyper-parameters for in-loss-miner may be investigated to reach the state-of-the-art performance scores on the benchmark datasets.

REFERENCES

- [1] A. Agarwala, J. Pennington, Y. Dauphin, and S. Schoenholz. Temperature check: theory and practice for training models with softmax-cross-entropy losses. *arXiv preprint arXiv:2010.07344*, 2020.
- [2] J. Bromley, I. Guyon, Y. Lecun, E. Sickinger, R. Shah, A. Bell, and L. Holmdel. Signature Verification using a Siamese Time Delay Neural Network. *Int. J. Pattern Recognit. Artif. Intell.*, 7(04):669—688, 1993.
- [3] D. Dayal Mohan, N. Sankaran, D. Fedorishin, S. Setlur, and V. Govindaraju. Moving in the Right Direction: A Regularization for Deep Metric Learning. In *Comput. Vis. Pattern Recognit.*, pages 14591–14599. IEEE Computer Society, 2020.
- [4] C. de Coulomb. Second mémoire sur l’électricité et le magnétisme. *Histoire de l’Académie Royale des Sciences*, page 578–611, 1785.
- [5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [6] Y. Duan, J. Lu, W. Zheng, and J. Zhou. Deep adversarial metric learning. *IEEE Transactions on Image Processing*, 29:2037–2051, 2020.
- [7] W. Ge, W. Huang, D. Dong, and M. R. Scott. Deep Metric Learning with Hierarchical Triplet Loss. In *Eur. Conf. Comput. Vis.*, volume 11210 LNCS, pages 272–288. Springer Verlag, 2018.
- [8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [9] P. M. D. Gray, T. Eavis, A. Inselberg, P. Valduriez, P. Valduriez, G. Graefe, G. Graefe, H. Zeller, G. Graefe, E. Pacitti, C. Koch, R. Zhang, B. Kemme, Y. Wu, H. Cheng, J. Han, P. Cudré-Mauroux, W. Galuba, S. Girdzijauskas, P. Felber, E. Biersack, A. Kementsietsidis, P. Triantafillou, I. Aekaterinidis, A. Datta, W. Galuba, S. Girdzijauskas, G. Weikum, A. Thomasian, P. Bonnet, D. Shasha, N. A. Lorentzos, J.-R. Wen, Z. Dou, R. Song, W. M. P. van der Aalst, C. Dyreson, S. S. Lightstone, P. Bonnet, D. Shasha, R. Johnson, E. Pitoura, K. Wada, D. Toman, G. Karabatis, C. Sirangelo, G. Grahne, J. Domingo-Ferrer, E. Zhang, Y. Zhang, B. Carterette, N. Craswell, N. Craswell, J. Kamps, Y. Manolopoulos, Y. Theodoridis, V. J. Tsotras, H. T. Shen, P. C. K. Hung,

- V. S. Y. Cheng, C. Clifton, P. C. K. Hung, Y. Zheng, S. Chow, S. Fischer-Hübner, C. Clifton, D. Suciu, T. Roelleke, J. Wang, S. Robertson, R. Cheng, J. Chen, V. S. Subrahmanian, B. He, D. Hiemstra, N. Palmer, W. M. P. van der Aalst, D. Ardagna, P. Helland, G. Ramírez, A. Trotman, P. Boncz, C. Sirangelo, K. Pinel-Sauvagnat, W.-C. Tan, S. Cohen-Boulakia, W.-C. Tan, S. Fischer-Hübner, H.-A. Jacobsen, Y. Diao, M. J. Franklin, D. Maier, and P. A. Tucker. Precision at n. In *Encycl. Database Syst.*, pages 2127–2128. Springer US, 2009.
- [10] R. Grosse. Lecture 9: Generalization.
- [11] B. Harwood, B. G. Vijay Kumar, G. Carneiro, I. Reid, and T. Drummond. Smart Mining for Deep Metric Learning. In *Int. Conf. Comput. Vis.*, volume 2017-October, pages 2840–2848. Institute of Electrical and Electronics Engineers Inc., 2017.
- [12] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Comput. Vis. Pattern Recognit.*, volume 2016-December, pages 770–778. IEEE Computer Society, dec 2016.
- [14] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *Int. Work. Similarity-Based Pattern Recognit.*, volume 9370, pages 84–92. Springer Verlag, 2015.
- [15] M. Kaya and H. S. Bilge. Deep Metric Learning: A Survey. *Symmetry (Basel)*, 11(9):1066, aug 2019.
- [16] S. Kim, D. Kim, M. Cho, and S. Kwak. Proxy Anchor Loss for Deep Metric Learning. In *Comput. Vis. Pattern Recognit.*, pages 3235–3244. Institute of Electrical and Electronics Engineers (IEEE), mar 2020.
- [17] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D object representations for fine-grained categorization. In *Int. Conf. Comput. Vis.*, pages 554–561. Institute of Electrical and Electronics Engineers Inc., 2013.
- [18] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin. A novel performance evaluation methodology for single-target trackers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(11):2137–2155, Nov 2016.
- [19] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research). 2010.
- [20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Neural Inf. Process. Syst.*, volume 25, pages 1097–1105, 2012.

- [21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Comput.*, 1(4):541–551, dec 1989.
- [22] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.
- [23] A. Machanavajjhala, J. Gehrke, M. M. Moro, A. Marian, R. Schenkel, M. Theobald, P. Baumann, T. Brinkhoff, J. Hansson, M. Xiong, E. Zhang, Y. Zhang, P.-N. Tan, J. Domingo-Ferrer, A. Arasu, J. Domingo-Ferrer, K. Wada, C. Patel, C. Weng, M. Young-Lai, R. Sion, V. Tannen, V. Tannen, D. W. Eembley, M. Lalmas, C. S. Jensen, R. T. Snodgrass, J. Pehcevski, B. Larsen, B. He, X.-J. Wang, L. Zhang, O. Vechtomova, R. Jimenez-Peris, M. Patiño-Martinez, A. Fekete, B. Kemme, K. Wada, F. Pedone, B. Kemme, R. Jiménez-Peris, M. Patiño-Martínez, R. Jimenez-Peris, M. Patiño-Martinez, Z. Despotovic, A. Gokhale, V. Novák, D. Zhang, Y. Du, M. Weiss, V. Christophides, G. Janée, C. Plaisant, O. Etzion, D. Papadias, Y. Tao, C. Li, A. Gokhale, C. Shahabi, B. He, Y. Zhang, J. B. Joshi, N. Craswell, A. N. Papadopoulos, A. Corral, A. Nanopoulos, Y. Theodoridis, and A. K. H. Tung. Recall. In *Encycl. Database Syst.*, pages 2348–2348. Springer US, Boston, MA, 2009.
- [24] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.
- [25] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No Fuss Distance Metric Learning using Proxies. In *Int. Conf. Comput. Vis.*, volume 2017-October, pages 360–368. Institute of Electrical and Electronics Engineers Inc., 2017.
- [26] K. Musgrave, S. Belongie, and S.-N. Lim. A Metric Learning Reality Check. In *Eur. Conf. Comput. Vis.*, 2020.
- [27] 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 H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.
- [28] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. In *Comput. Vis. Pattern Recognit.*, volume 07-12-June, pages 815–823. IEEE Computer Society, 2015.
- [29] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 118–126, 2015.

- [30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Int. Conf. Learn. Represent. International Conference on Learning Representations, ICLR*, sep 2015.
- [31] K. Sohn. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In *Neural Inf. Process. Syst.*, volume 29, pages 1857–1865, 2016.
- [32] K. Sohn. Improved Deep Metric Learning with Multi-class N-pair Loss Objective. In *Neural Inf. Process. Syst.*, volume 29, pages 1857–1865, 2016.
- [33] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep Metric Learning via Lifted Structured Feature Embedding. *Comput. Vis. Pattern Recognit.*, 2016-Decem:4004–4012, nov 2015.
- [34] I. W. Stewart. Advanced classical mechanics. *Edited by Francisco Machado, Thanawuth Thanathibodee, and Prashanth S. Venkataram. https://ocw.mit.edu/courses/physics/8-09-classical-mechanics-iii-fall-2014/lecturenotes/MIT8_09F14_full.pdf Downloaded, 11(10):2020*, 2016.
- [35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Comput. Vis. Pattern Recognit.*, volume 07-12-June-2015, pages 1–9. IEEE Computer Society, oct 2015.
- [36] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu. Learning Fine-grained Image Similarity with Deep Ranking. In *Comput. Vis. Pattern Recognit.*, pages 1386–1393. IEEE Computer Society, apr 2014.
- [37] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin. Deep Metric Learning with Angular Loss. In *Int. Conf. Comput. Vis.*, volume 2017-Octob, pages 2612–2620. Institute of Electrical and Electronics Engineers Inc., 2017.
- [38] Y. Wang and Z. Wang. A survey of recent work on fine-grained image classification techniques. *Journal of Visual Communication and Image Representation*, 59:210–214, 2019.
- [39] X.-S. Wei, J. Wu, and Q. Cui. Deep Learning for Fine-Grained Image Analysis: A Survey, 2019.
- [40] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical report, California Institute of Technology, 2010.
- [41] P. Wohlhart and V. Lepetit. Learning Descriptors for Object Recognition and 3D Pose Estimation. In *Comput. Vis. Pattern Recognit.*, volume 07-12-June-2015, pages 3109–3118. IEEE Computer Society, feb 2015.

- [42] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl. Sampling Matters in Deep Embedding Learning. In *Int. Conf. Comput. Vis.*, volume 2017-Octob, pages 2859–2867. Institute of Electrical and Electronics Engineers Inc., jun 2017.
- [43] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Eur. Conf. Comput. Vis.*, volume 8689 LNCS, pages 818–833. Springer Verlag, 2014.
- [44] B. Zhao, J. Feng, X. Wu, and S. Yan. A survey on deep learning-based fine-grained object classification and semantic segmentation. *International Journal of Automation and Computing*, 14(2):119–135, 2017.
- [45] M. Zheng, Q. Li, Y.-a. Geng, H. Yu, J. Wang, J. Gan, and W. Xue. A survey of fine-grained image categorization. In *2018 14th IEEE International Conference on Signal Processing (ICSP)*, pages 533–538. IEEE, 2018.
- [46] W. Zheng, Z. Chen, J. Lu, and J. Zhou. Hardness-Aware Deep Metric Learning. In *Comput. Vis. Pattern Recognit.*, 2019.

APPENDIX A

CLASS STRUCTURE OF OUR FRAMEWORK

We implement our framework with the help of PyTorch library [27] on Python. The generic classes of the framework and their interactions are shown in Figure A.1 as a class diagram.

A.1 General Structure

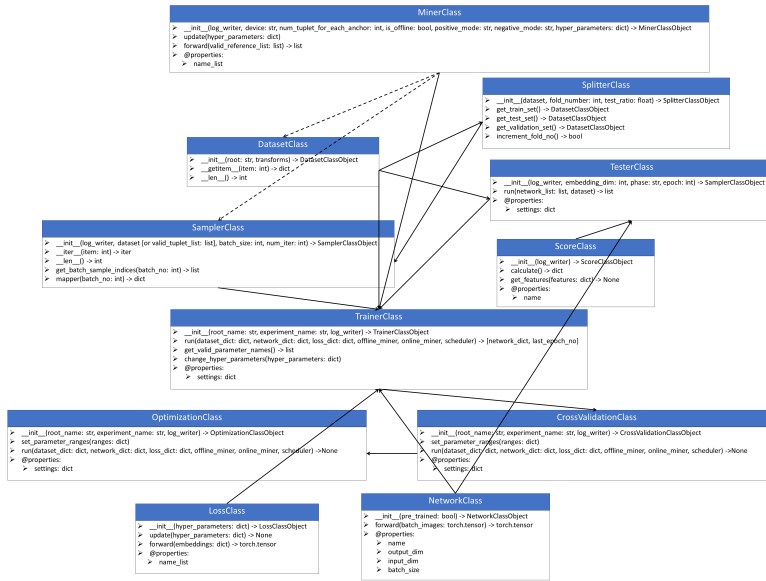


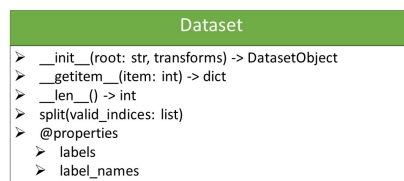
Figure A.1: The figure about the relationship between the class objects in the implemented framework.

We coded the framework with the help of the inheritance property of the object-oriented programming paradigm. To illustrate, the generic dataset class inherits CUB200-2011 class, and the generic score class inherits the *Recall@K*, and *Precision@R* score

classes. Thus, we show the public methods and variables of these classes in order to provide a comprehensive explanation of our framework.

A.2 Public Functions and Properties of the Classes

A.2.1 Dataset Class



Dataset
> <code>__init__(root: str, transforms) -> DatasetObject</code>
> <code>__getitem__(item: int) -> dict</code>
> <code>__len__() -> int</code>
> <code>split(valid_indices: list)</code>
> <code>@properties</code>
> <code>labels</code>
> <code>label_names</code>

Figure A.2: The public methods and variables of dataset class objects.

This class is inherited from the original *Dataset* class in PyTorch. The functions and the public variables in this class is shown in Figure A.2 The standard functions `__getitem__()` and `__len__()` are used for the data-loader object in the training and testing phases. The `split()` function is implemented so that the splitter splits dataset into 3 parts: Train, validation and test sets. The public variables are `labels` and `label_names`. By using these variables, we test the dataset objects if they are implemented correctly or not. Moreover, the `labels` variable is utilized in the sampler objects to set all the batch of training.

A.2.2 Transform Class

This class returns the list of transformation methods in the `compose()` function of PyTorch. By using this class, we can set the data augmentation methods utilized in the experiment we performed.

A.2.3 Splitter Class

Splitter	
>	<code>__init__(dataset, total_fold: int, fold_no: int, test_ratio: float) -> SplitterObject</code>
>	<code>increment_fold_no(fold_no: int) -> bool</code>
>	<code>reset_fold_no(fold_no: int) -> None</code>
>	<code>@properties</code>
>	<code>train_set -> DatasetObject</code>
>	<code>test_set -> DatasetObject</code>
>	<code>validation_set -> DatasetObject</code>
>	<code>fold_no -> int</code>

Figure A.3: The public methods and variables of splitter class objects.

This class is exploited for separated the dataset objects into the train, validation, and test sets. There are mainly two types of splitter: The first one is for the close-set. It separates the test and train+validation sets at first. After that, the train and validation sets are split. The classes in these sets are not disjoint, whereas the samples in them are. On the other hand, the second one is called the open set splitter. It imitates the zero-shot scenario. Therefore, both the classes and samples are disjoint in these sets. Note that when the open set splitter is used, the classes of train and validation set are also disjoint. The public functions and variables are shown in Figure A.3. The purpose of usage of this class is to get the dataset objects by using the variables `train_set`, `validation_set`, and `test_set`. Also, the different folds of the cross-validation can be obtained by changing `fold_no`.

A.2.4 Sampler Class

Sampler	
>	<code>__init__(log_writer, batch_size: int, num_iter: int) -> SamplerObject</code>
>	<code>__iter__(item: int) -> iter</code>
>	<code>__len__() -> int</code>
>	<code>set(dataset, miner, **kwargs) -> list</code>
>	<code>mapper(embeddings, batch_no: int) -> dict</code>

Figure A.4: The public methods and variables of sampler class objects.

A sampler object is exploited for adjusting the batch during the training phase. There are mainly two types of samplers: Online and offline. The online sampler does not consider the distances of the samples in the embedding space; therefore, there is no need for extraction of embeddings. It only considers the `labels` variable of the dataset object. After constructing the batch, the tuples are selecting with the help of the miner object. On the other hand, the offline sampler takes the distance matrix of the embeddings in the account. Thus, the extraction of the embeddings is needed before constructing the batches. Clearly, the online sampler is faster than the offline one. The public methods and the properties of the sampler object are shown in Figure A.4. The `set()` function is used before the training phase, while the `mapper()` function is utilized in the training method in order to select the tuples from the embeddings extracted by the network.

A.2.5 Miner Class

```

Miner
┆ __init__(log_writer, device: str) -> MinerObject
┆ set(**kwargs) -> None
┆ update(hyper_parameters: dict) -> None
┆ mine(features: dict) -> list
┆ @properties:
┆   argument_names: list
┆   device: str
┆   hyper_parameters: dict

```

Figure A.5: The public methods and variables of miner class objects.

The usage of a miner object is to select the most desirable tuples from the batch embeddings. It may also use for mining all the samples in the dataset; however, this calculation needs extremely high resources and computational time. The generic methods and the variables are shown in the Figure A.5.

A.2.6 Network Class

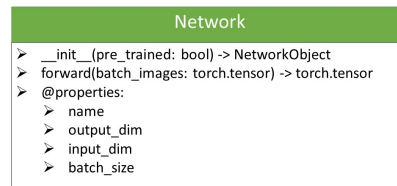


Figure A.6: The public methods and variables of network class objects.

The network class is inherited from the `nn.Module` class of PyTorch. There are some more properties named `name`, `output_dim`, `input_dim`, and `batch_size` as shown in Figure A.6. The `forward` method extracts the batch of the embeddings of the images while calculating the gradients of backpropagation.

A.2.7 Loss Class

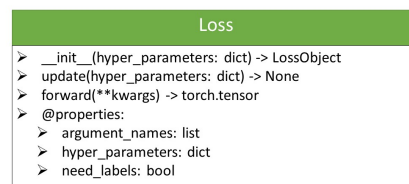


Figure A.7: The public methods and variables of loss class objects.

The loss class is inherited from the `nn.Module` class of PyTorch as well, like the network class; hence, the `forward()` function is used for calculating the loss value of the current batch in the training phase. Also, the `update()` function is utilized for changing the hyper-parameter of the loss function. The class diagram is shown in Figure A.7

A.2.8 Trainer Class

Trainer	
➤	<code>__init__(experiment_name: str, root_name: str, log_writer) -> TrainerObject</code>
➤	<code>set(dataset_dict: dict, network_dict: dict, loss_dict: dict, sampler, minier, optimizer, scheduler) -> None</code>
➤	<code>run(stop_score_name: str, hyper_parameters: dict) -> (network_dict: dict, final_epochs: int, best_validation_score: float): dict</code>
➤	<code>update_hyper_parameters(hyper_parameters: dict) -> None</code>
➤	@properties:
➤	settings: dict
➤	hyper_parameters: dict

Figure A.8: The public methods and variables of trainer class objects.

The trainer objects are created from a glue class that trains the network using certain objects. First, the objects needed for the experiment are created. These objects are set by using the `set()` function in Figure A.8. After that, the hyper-parameters are determined; otherwise, the trainer object trains the network with the default values. Finally, the `run()` method is called for training the network with the train set. The data in the validation set is also given to this object in order to determine if the experiment should be stopped or not.

A.2.9 Log Writer Class

The logger is frequently utilized objects during training in order to debug the codes. We implement a standard logger with the help of the `logging`, `tqdm` and `tensorboard` libraries. It is used as an input for almost all of the classes. This logger class is also able to show the results on the plots in `tensorboard` format.

A.2.10 Score Class

Score
> <code>__init__(log_writer) -> ScoreObject</code>
> <code>calculate(features: dict) -> dict</code>
> @properties:
> <code>log_writer</code>

Figure A.9: The public methods and variables of score class objects.

The score classes are implemented as seen in Figure A.9. There is only one public method called `calculate()`. This function takes the features as the dictionary format and returns the score values from them.

A.2.11 Tester Class

Tester
> <code>__init__(scores_dict: dict, log_writer, phase: str, epoch: int) -> TesterObject</code>
> <code>run(network_list: list, dataset, log_writer, phase: str, epoch: int) -> dict</code>
> @properties:
> <code>settings: dict</code>

Figure A.10: The public methods and variables of tester class objects.

Like the trainer class, the tester class is another glue class that is used for calculating the validation or test scores of the given network. Any tester object has fixed score objects created in itself; therefore, the score object is not taken by a tester object. However, the phase name, epoch number, and the logger objects should be given as seen in Figure A.10.

A.2.12 Experiment Class

The experiment class is used for performing the experiments with different hyper-parameters. There are several children class of it. The first one is named as `SingleExperiment`. It is run with the specified hyper-parameters given before the experiment. Secondly, the `CrossValidation` uses the specified hyper-parameters, again; but it trains the network for all the folds of the validation sets and calculates the test score of the best network.

Moreover, hyper-parameter optimization is also needed for comparing the methods in the literature in a fair manner. Therefore, we implement the `GridSearch`, `RandomSearch`, `BayesianSearch` child classes for this purpose as mentioned in Section 1.4.

APPENDIX B

DISCUSSION ON THE FIRST MARGIN AND THE HINGE FUNCTION FOR $R = 0$ CASE

When $r = 0$, the boundary condition becomes as Equation (B.1),

$$\|f_a - f_n\|_2 + \|f_p - f_n\|_2 = 2 \cdot r_{major}, \quad (\text{B.1})$$

which is the equation of a hyper-ellipsoid, r_{major} is the semi-major axis (that is depend on the values of s , m & ρ), and f_a & f_p are the foci of this hyper-ellipsoid. The distance between the center and one of the foci (named as linear eccentricity) of this manifold is $\|f_a - f_p\|_2/2$. The semi-minor axis are equal to $r_{minor} = \sqrt{r_{major}^2 - \|f_a - f_p\|_2^2/4}$.

In this section, starting from Equation (3.20), we aim to find the optimal m_1 that produces a hyper-ellipsoid that is large enough. Obviously, the next question arises about the size limits of such a hyper-ellipsoid. At this point, we consider a filtered region of the ranking loss function. This region can be drawn as follows: It is a hyper-sphere whose center is at the location of anchor, f_a . A positive (which is at f_p) instance is at the surface of this manifold (hyper-sphere) if the margin is neglected. It means the radius of this hyper-sphere holds $r = \|f_a - f_p\|_2$ if $m_{ranking} = 0$. If the margin $m_{ranking}$ is also taken into consideration, the radius of this hyper-sphere is changed from r to $r + m_{ranking}$.

Let us consider the case in which our ellipsoid contains the two hyper-spheres; the center of one hyper-sphere is at the location of the anchor, and that of the other is at the location of the positive instance. To find this ellipsoid whose hyper-volume is the smallest, we should solve a convex optimization problem. Basically, we reduce the dimension to 2 and find the ellipse that has a minimum area (its area is equal

to $\pi \cdot r_{major} \cdot r_{minor}$), and that contains the two circles in Equation (B.2) when the distance between the anchor and the positive instance is equal to d .

$$\begin{aligned} & \min_{a,b} \pi \cdot \sqrt{a \cdot b} \text{ such that} \\ & \left(x - \frac{d}{2}\right)^2 + y^2 = (d + m_{ranking})^2 \text{ and} \\ & \frac{x^2}{a} + \frac{y^2}{b} = 1 \text{ have only} \\ & \text{two intersection points at the same } x = x_0. \end{aligned} \tag{B.2}$$

For solving this problem, first, we find the x-axis of the intersection points. It becomes a quadratic equation shown in Equation (B.3) with the discriminant for x , which should be 0 because the roots should be x_0 and not be distinct.

$$\begin{aligned} & \left(1 - \frac{b}{a}\right) \cdot x^2 - d \cdot x + b - (d + m_{ranking})^2 + \frac{d^2}{4} = 0 \\ & \Delta = d^2 - 4 \cdot \left(1 - \frac{b}{a}\right) \cdot \left(b - (d + m_{ranking})^2 + \frac{d^2}{4}\right) = 0. \end{aligned} \tag{B.3}$$

Then, we can find a relationship between a and b shown in Equation (B.4), by defining the two functions of b ; $\eta_1(\cdot)$, and $\eta_2(\cdot)$.

$$\hat{a} = \hat{b} \cdot \left(1 + \frac{d^2/4}{\hat{b} - (d + m_{ranking})^2}\right) = \frac{\eta_1(\hat{b})}{\eta_2(\hat{b})}. \tag{B.4}$$

Note that the optimization problem is convex; hence, we can minimize the $\sqrt{a \cdot b}$ by using the condition $\eta_1(b) \cdot \frac{\partial \eta_2(b)}{\partial b} = \eta_2(b) \cdot \frac{\partial \eta_1(b)}{\partial b}$. The solution of this equality is shown in Equation (B.5).

$$\hat{b} = (d + m_{ranking})^2 + \frac{d \cdot \sqrt{d^2 + 32 \cdot (d + m_{ranking})^2} - d^2}{16}. \tag{B.5}$$

We cannot adjust b without changing the a since the equation $b^2 + \frac{d^2}{4} = a^2$ always holds. Hence, the semi-major axis should be satisfied the conditions $r_{major}^2 \geq \frac{\hat{a}}{4}$ and $r_{major}^2 \geq \frac{\hat{b}}{4} + \frac{d^2}{16}$. Moreover, from Equation (B.1) and (3.20), we can determine the

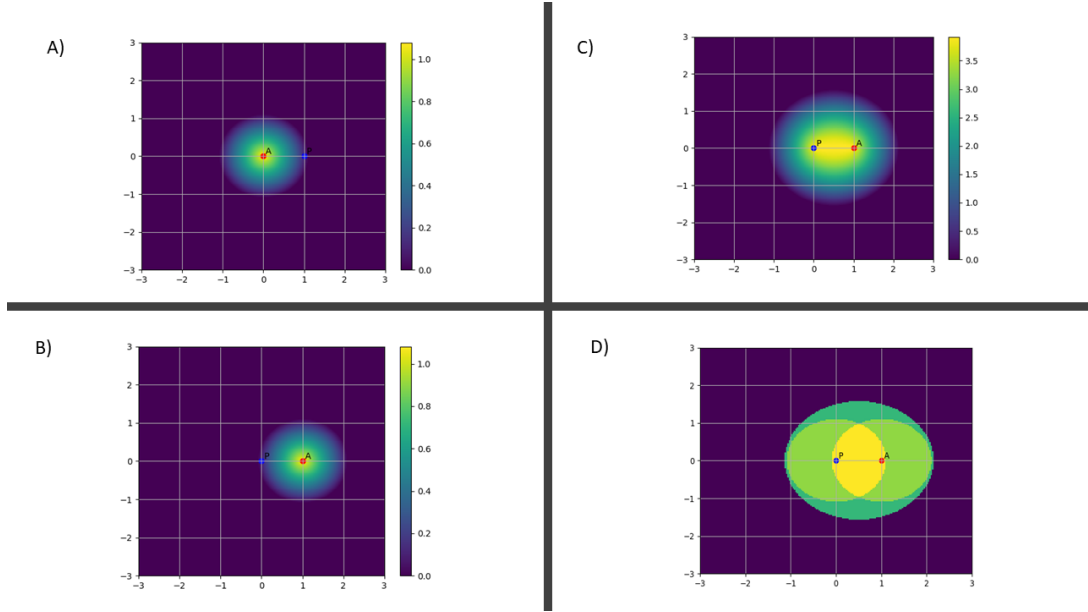


Figure B.1: The 2-dimensional representation of the loss function values when the anchor is located in $(0.0, 0.0)$ and the positive is at $(1.0, 0.0)$. The plot (A) shows the circle due to the first object, which is selected as the anchor of the triplet. Similarly, plot (B) represents the region when the object at $(1.0, 0.0)$ is selected as the anchor point. The plot (C), on the other hand, shows the loss function values for the our loss function when $r = 0, s = 0, d = 1, \rho = 12/7, m_1 = 4.6381$. Finally, the plot (D) shows the overlapping area of the previous three plots.

$m_{proposed}$ value as in Equation (B.6), where the $d = \|f_a - f_p\|_2$ value can be taken as the mean value of intra-class distance.

$$r_{major} = \frac{1 - r}{2 \cdot (s + 1) \cdot \rho} \cdot d^{s+1} + \frac{1 - r}{2 \cdot \rho} \cdot m_{proposed}. \quad (\text{B.6})$$

To give an example, when the case is $r = 0, s = 0, d = 1, \rho = 12/7, m_{ranking} = 0.1$, the inequality $m_1 \geq 4.6381$ should be satisfied for non-trivial training process. The 2 dimensional plot of the filtered regions for ellipse and the circles are shown in Figure B.1 when $m_1 = 4.6381$.