Contrastive Learning for Visual Object Recognition

Raquel Filipa Lebre de Abreu

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

(2º ciclo de estudos)

Orientador: Professor Doutor João Carlos Raposo Neves

Co-orientador: Professor Doutor Vasco Ferrinho Lopes

Covilhã, junho de 2025

Declaração de Integridade

Eu, Raquel Filipa Lebre de Abreu, que abaixo assino, estudante com o número de inscrição M12834 de Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 09/06/2025

Assinado por: **Raquel Filipa Lebre de Abreu**
Num. de Identificação: 15960829
Data: 2025.06.09 17:09:23+01'00'

Dedication

Dedicated to my grandparents.

Acknowledgements

This dissertation would not have been possible without the help and support of some people to whom I am eternally grateful.

To Professor João Neves for his guidance and help throughout this process.

To my parents and sister, Paulo, Paula and Joana, who always encouraged me to work hard to achieve what I wanted and for all their support.

To my boyfriend, Bernardo, who has been on this journey with me since day one, has been with me through the ups and downs, has helped me, supported me, and has been my safe place.

To my friends who have accompanied me throughout this journey, especially Lara, who, even from afar, have always helped me and encouraged me to keep going and give it my all.

To everyone who contributed directly and indirectly to completing this work,

Thank you.

Resumo

Contrastive Learning emergiu como uma técnica fundamental na aprendizagem automática, especialmente no contexto da aprendizagem auto-supervisionada. Ao utilizar objetivos contrastivos, estes modelos aprendem a aproximar representações de pontos de dados semelhantes no espaço latente, ao mesmo tempo que afastam aqueles distintos, promovendo uma discriminação eficiente entre padrões. Esta abordagem tem demonstrado um impacto significativo em diversas áreas, incluindo visão computacional e processamento de linguagem natural. Esta dissertação investiga o tema *Contrastive Learning*, com foco na reconhecimento visual de objetos. Especificamente, são explorados os principais métodos contrastivos, com o objetivo de identificar limitações que possam ser abordadas e melhoradas. Várias soluções foram propostas com o objetivo de superar as limitações encontradas, que serão exploradas ao longo deste documento. Os resultados obtidos variaram em termos de desempenho: algumas das soluções apresentaram desempenhos inferiores em comparação com os resultados originais, enquanto outras se aproximaram bastante dos resultados iniciais, mas não os ultrapassando. Contudo, uma das soluções demonstrou desempenho superior aos métodos base, demonstrando a possibilidadede melhorias adicionais através de ajustes e melhorias contínuas.

Palavras-chave

Inteligência Artificial, *Contrastive Learning*, Visão Computacional

Resumo alargado

O reconhecimento visual de objetos é um aspeto fundamental da Visão Computacional, com aplicações numa série de áreas, desde veículos autónomos a sistemas de recuperação de imagens. A procura de sistemas de reconhecimento visual eficientes está a crescer, o que está a levar ao desenvolvimento de novas metodologias para melhorar o desempenho destes sistemas. Entre estas metodologias, uma área que se tem demonstrado muito promissora é a área de *Contrastive Learning*. Baseado na premissa de contrastar amostras entre si, este método tem tido resultados muito positivos no que diz respeito à aprendizagem de boas representações visuais.

Os métodos de reconhecimento visual de objetos enfrentam muitas vezes desafios na captura de características com nuances e no tratamento de conjuntos de dados em grande escala com diferentes complexidades visuais. *Contrastive Learning* oferece uma nova perspetiva ao enfatizar a discriminação entre pares positivos e negativos dentro do conjunto de dados. Esta abordagem visa aprender um espaço de representação em que as instâncias semelhantes são aproximadas e as instâncias que não o são são afastadas, facilitando assim a criação de caraterísticas visuais mais discriminativas e robustas.

Contrastive Learning tem uma história rica, com contribuições de vários investigadores e desenvolvimentos ao longo dos anos. As principais áreas de estudo neste domínio desenvolvem o estudo das funções de perda, quer adicionando exemplos negativos à definição inicial, quer considerando mais do que um exemplo positivo, sobre o estudo da importância da estratégia de amostragem e também sobre a estrutura da própria rede, mas os exemplos mais recentes desenvolvem a aprendizagem não supervisionada, que provou ser bastante eficiente. Estes serão discutidos com mais pormenor mais adiante nesta dissertação.

Esta dissertação tem como objetivo aprofundar o campo da *Contrastive Learning* para o problema do reconhecimento visual de objetos. Para tal, será realizada uma avaliação dos métodos existentes, de forma a perceber as suas principais limitações. Idealmente, será proposta uma solução para ultrapassar alguma limitação, de forma a tentar melhorar, ou mesmo ultrapassar, o estado da arte existente. Este esforço de investigação é motivado pela crença de que representações visuais melhoradas podem levar a sistemas de reconhecimento de objetos mais precisos e eficientes.

Em suma, esta dissertação de mestrado espera contribuir para dar um passo em frente no campo do reconhecimento visual de objetos, tirando partido dos excelentes resultados obtidos até agora pela área de *Contrastive Learning*. Ao tentar compreender as limitações dos

métodos atuais e ultrapassá-las, esta investigação pretende encontrar informação que poderá ser importante no futuro do reconhecimento visual de objetos e, consequentemente, da visão computacional.

O âmbito desta dissertação enquadra-se na área de Inteligência Artificial e Visão Computacional, uma vez que *Contrastive Learning* é uma técnica utilizada em tarefas de visão que utiliza o conceito de comparação de amostras entre si, com o objetivo de compreender tanto os atributos partilhados entre diferentes classes de dados como as caraterísticas distintivas que diferenciam uma classe de dados de outra.

O trabalho a desenvolver com esta dissertação tem como objetivo fazer avançar o estado da arte de *Contrastive Learning* para a obtenção de descritores visuais robustos para problemas gerais de reconhecimento de objetos. Para tal, prevê-se uma revisão dos métodos do estado da arte, bem como a implementação e avaliação dos métodos mais relevantes sobre o tema. Espera-se que a análise das falhas destas abordagens evidencie algumas das fraquezas da estratégia de aprendizagem e abra caminho à proposta de uma nova estratégia para melhorar a discriminabilidade dos descritores visuais aprendidos.

Tendo tudo em conta, os objetivos definidos para esta dissertação são os seguintes:

- Analisar em pormenor os métodos existentes, e as suas principais contribuições. Estes métodos devem ser implementados e avaliados de forma a comprovar a sua eficiência, e também para determinar as suas limitações;
- Com base nas limitações dos métodos, este trabalho pretende desenvolver uma nova estratégia para colmatar as falhas das abordagens existentes;
- Comparar a estratégia proposta em conjuntos de dados padrão de reconhecimento de objetos para demonstrar as suas vantagens quando comparada com o estado da arte sobre o tema.

A abordagem adotada para atingir os objetivos acima definidos foi a seguinte:

1. Analisar o estado da arte dos métodos de *Contrastive Learning*;
2. Implementar os métodos analisados no passo anterior, tendo em conta os requisitos de *hardware* e *software* necessários para cada um dos métodos;
3. Analisar os métodos implementados com mais pormenor, para tentar encontrar limitações nesses métodos e também para desenvolver uma nova estratégia para melhorar os resultados com base no método existente;

4. Implementar as diferentes modificações propostas;
5. Analisar os resultados da métrica de avaliação das modificações, *Accuracy* 1, e também criar gráficos complementares que ajudem a visualizar melhor as modificações implementadas.

Este documento está organizado em seis capítulos. O conteúdo e a organização dos capítulos podem ser resumidos da seguinte forma:

- Capítulo 1 - Introdução - introduz o tema desta dissertação. Na primeira secção descreve-se o âmbito e a motivação desta dissertação, seguindo-se a descrição do problema abordado e dos objetivos. A abordagem adoptada para a resolução do problema é também descrita e, em seguida, são analisados os principais contributos e resultados deste estudo. Finalmente, apresenta-se a organização do documento;
- Capítulo 2 - Estado da Arte - apresenta os conceitos fundamentais para uma melhor compreensão do tema abordado por esta proposta. É apresentada uma breve introdução à Aprendizagem Contrastiva, seguida de terminologia e conceitos preliminares. Trabalhos relacionados com este tema são também apresentados no capítulo;
- Capítulo 3 - Metodologia - explica o problema desta dissertação, seguido das soluções propostas para o resolver;
- Capítulo 4 - Detalhes da Implementação - apresenta o método implementado, começando por analisar o conjunto de dados utilizado, os diferentes candidatos a métodos a implementar e alguns desafios encontrados e mais detalhes do método implementado. De seguida são descritas as diferentes soluções para tentar resolver as lacunas encontradas;
- Capítulo 5 - Resultados - apresenta os resultados preliminares e os resultados das modificações implementadas, comparando-os e analisando-os;
- Capítulo 6 - Conclusão e Trabalho Futuro - descreve as principais conclusões desta dissertação e apresenta também o trabalho a desenvolver no futuro.

Abstract

Contrastive Learning has emerged as a key technique in machine learning, especially in the context of self-supervised learning. By using contrastive objectives, these models learn to approximate representations of similar data points in latent space, while distancing those that are distinct, promoting efficient discrimination between patterns. This approach has demonstrated a significant impact in several areas, including computer vision and natural language processing. This dissertation investigates the topic of Contrastive Learning, with a focus on visual object recognition. Specifically, the main contrastive methods are explored, with the aim of identifying limitations that can be addressed and improved. Various solutions have been proposed with the aim of overcoming the limitations found, which will be explored throughout this document. The results obtained varied in terms of performance: some of the solutions showed lower performance compared to the original results, while others came very close to the initial results, but did not surpass them. However, one of the solutions outperformed the base methods, demonstrating the possibility of further improvements through continuous adjustments and improvements.

Keywords

Artificial Intelligence, Contrastive Learning, Computer Vision

Contents

List of Figures

Acronyms

ACM	Association for Computing Machinery
AI	Artificial Intelligence
AMDIM	Augmented Multiscale DIM
BYOL	Bootstrap Your Own Latent
CCS	Computing Classification System
CL	Contrastive Learning
CLSA	Contrastive Learning with Stronger Augmentations
CNN	Convolutional Neural Network
CV	Computer Vision
DCL	Decoupled Contrastive Learning
DIM	Deep InfoMax
GPaCo	Generalized Parametric Contrastive Learning
InfoNCE	InfoMax Noise Contrastive Estimation
K-NN	k Nearest Neighbors
LA	Local Aggregation
LooC	Leave-one-out Contrastive Learning
ML	Machine Learning
MoCo	Momentum Contrast
NCE	Noise Contrastive Estimation
NLP	Natural Language Processing
NN	Neural Network
NPC	Negative-Positive-Coupling
NT-Xent	Normalized Temperature-scaled cross-entropy
PaCo	Parametric Contrastive Learning
PCL	Prototypical Contrastive Learning
ReID	Person Re-Identification
RINCE	Robust InfoNCE

SelfCon Self-Contrastive Learning

SGD Stochastic Gradient Descent

SimCLR Simple Contrastive Learning Representation

ST-DIM Spatiotemporal Deep InfoMax

SupCon Supervised Contrastive Loss

SwAV Swapping Assignments between multiple Views of the same image

TCN Time Contrastive Network

VDIM Video Deep InfoMax

VINCE Video Noise Contrastive Estimation

Chapter 1

Introduction

This dissertation was produced in the context to the Master's program in Computer Science and Engineering at University of Beira Interior, as part of the requirements to obtain the degree. The underlying work described herein was performed in the 2023/24 and 2024/25 academic years.

This chapter provides an introduction to the subject of the dissertation, starting with the motivation and the scope of the work, followed by the addressed problem and the main objectives to be achieved with this project. Next, it is described the adopted approach for solving the problem. The chapter ends with a description of the organization of the document.

1.1 Scope and Motivation

1.1.1 Motivation

Visual object recognition is a fundamental aspect of Computer Vision (CV), with applications in a wide range of areas, from autonomous vehicles ([1], [2]) to image retrieval systems ([3], [4]). The demand for efficient visual recognition systems is growing, and this is leading to the development of new methodologies to improve the performance of these systems. Among these methodologies, one area that has shown great promise is **Contrastive Learning (CL)**. Based on the premise of contrasting samples with each other, this method has had very positive results when it comes to learning good visual representations.

Visual object recognition methods often face challenges in capturing nuanced features and dealing with large-scale datasets with diverse visual complexities. CL offers a new perspective by emphasizing discrimination between positive and negative pairs within the dataset. This approach aims to learn a representation space where similar instances are moved closer together and dissimilar instances are moved further apart, thus facilitating the creation of more discriminative and robust visual features ([5], [6], [7]).

CL has a rich history, with contributions from various researchers and developments over the years. The main areas of study in this area elaborate on the study of loss functions, either adding negative examples to the initial definition ([8], [9], [10]), or considering more than one positive example ([11]), on the study of the importance of the sampling strategy ([12])

and also on the structure of the network itself, but more recent examples elaborate on unsupervised learning, which has proven to be quite efficient ([13], [14], [15]). These will be discussed in more detail later in this dissertation.

This dissertation aims to delve deeper into the field of CL for the problem of visual object recognition. To this end, an evaluation of existing methods will be carried out in order to understand their main limitations. Ideally, a modification will be proposed to overcome some limitation, in order to try to improve, or even surpass, the existing state-of-the-art. This research effort is motivated by the belief that improved visual representations can lead to more accurate and efficient object recognition systems.

In summary, this dissertation hopes to contribute to take a step forward in the field of visual object recognition, taking advantage of the excellent results obtained so far by Contrastive Learning. By trying to understand the limitations of current methods and overcome them, this research aims to find information that may be important in the future of visual object recognition, and, therefore, computer vision.

1.1.2 Scope

The scope of this dissertation falls in the area of Artificial Intelligence (AI) and Computer Vision (CV), since CL is a technique used in vision tasks by leveraging the concept of comparing samples with each other, aiming to understand both the shared attributes among different data classes and the distinctive features that differentiate one data class from another. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System (CCS), a de facto standard for computer science, the scope of this master's project, reflected in this dissertation, is defined by the categories named:

- **Artificial Intelligence** → **Computer Vision**;
- *Artificial Intelligence* → *Computer Vision* → *Computer Vision Problems* → *Object Recognition*;
- *Machine Learning* → *Learning Paradigms*.

1.2 Addressed Problem and Objectives

1.2.1 Addressed Problem

The work to be developed with this dissertation aims to push forward the state-of-the-art on contrastive learning for obtaining robust visual descriptors for general object recognition problems. For this, a review of the state-of-the-art methods is envisaged, as well as the implementation and evaluation of the most relevant methods on the topic. The analysis of the failures of these approaches is expected to highlight some of the weakness in the learning strategy, and pave the way for proposing a novel strategy for improving the discriminability of the learned visual descriptors.

1.2.2 Objectives

Taking everything into account, the objectives set for this dissertation are as follows:

- Analyze in detail the existing methods, and their main contributions. These methods should be implemented and evaluated in order to prove their efficiency, and also to determine their limitations;
- Based on the limitations of the methods, this work intends to develop a novel strategy to address the shortcomings of existing approaches;
- Benchmark the proposed strategy on standard object recognition datasets to demonstrate its advantages when compared with the state-of-the-art on the topic.

1.3 Adopted Approach for Solving the Problem

The approach adopted to achieve the objectives defined above was as follows:

1. Analyze the state-of-the-art Contrastive Learning methods;
2. Implement the methods analyzed in previous step, taking into account the hardware and software requirements needed for each of the methods;
3. Analyze the implemented methods in more detail, to try to find limitations in these methods and also to develop a new strategy to improve the results based on the existing method;
4. Implement the different solutions proposed;

5. Analyze the results of the metric for evaluating the solutions, Accuracy 1, and also create complementary plots that help to better visualize the implemented solutions.

1.4 Document Organization

This document is organized into six chapters. The contents and organization of the chapters can be summarized as follows:

- Chapter 1 - Introduction - introduces the subject of this dissertation. Section 1.1 describes the scope and motivation of this dissertation, followed by a description of the addressed problem and the objectives (in section 1.2). The adopted approach for solving the problem is also described (section 1.3) and finally, the organization of the document is presented (present section);
- Chapter 2 - State of the Art - displays the fundamental concepts to better understand the topic approached by this proposal. A brief introduction to Contrastive Learning is presented, followed by terminology and preliminary concepts. Related work of this topic is also presented in the chapter;
- Chapter 3 - Methodology - explains the problem of this dissertation, followed by the solutions proposed to solve it;
- Chapter 4 - Implementation Details - presents the implemented method, starting by analyzing the dataset used (section 4.2), the different candidates for methods to implement and some challenges encountered (section 4.3) and further details of the implemented method (section 4.4). The different solutions to try to solve the gaps found are then described;
- Chapter 5 - Results - presents the preliminary results and the results of the modifications implemented, comparing and analyzing these;
- Chapter 6 - Conclusion and Future Work - describes the main conclusions of this dissertation and also presents the work to be done in the future.

Chapter 2

State of the Art

2.1 Introduction

This chapter defines some important concepts and terminology for this work, namely what contrastive learning is. Also, it introduces main works in this area, as well as their evolution over time. Even though CL has application in a multitude of problems, most of the related work presented ahead focuses on computer vision, as this is the main subject of this dissertation.

2.2 Terminology and Preliminary Concepts

In the realm of CV and Machine Learning (ML), the quest for robust and efficient methods to enable machines to understand visual information has been an everlasting challenge. Visual Object Recognition, a fundamental task in this domain, involves teaching machines to identify and categorize objects within images. As the field progresses, researchers have been exploring innovative approaches to enhance the learning process, and one such paradigm that has gained significant attention is Contrastive Learning (CL).

CL is a method designed to train models by emphasizing the differences between similar and dissimilar pairs of data samples. It makes the most of the differences and similarities between data, allowing similar instances to be brought together and those that are different to be moved apart in the latent space. CL does this by extracting meaningful representations by contrasting positive and negative pairs. This approach has already proven to be very efficient in various tasks in numerous domains, including Natural Language Processing (NLP) and CV.

As with other machine learning methods, CL can be separated into **supervised** and **unsupervised** approaches. In the first case, the data used to train the model is labeled; the models are trained specifically to differentiate between similar and dissimilar data. The objective here is the same as in CL, and has been mentioned before. In the second case, as the name suggests, the algorithm learns representations from unlabeled data ([16]). This approach takes advantage of pretext tasks, which create positive and negative pairs from the

data. These tasks are designed to encourage the model to pick up significant characteristics and similarities in the data. One of these tasks, that is often used in CV, is the generation of altered views of the same image - several images are created from an anchor image with data augmentation method that will serve as positive examples for the model, while instances from other images are negative pairs.

Although many contrastive learning methods initially focused on supervised settings - and with very positive results - self-supervised learning methods have been gaining more and more attention nowadays, and have had even more surprising results. This will be examined in more detail in the next section.

It is already known that the main objective of Contrastive Learning is to make similar instances come closer together while different instances are encouraged to be further apart. But how is this done in practice? One of the ingredients used is **data augmentation**, as mentioned earlier. Data augmentation refers to the process of creating new data by altering the existing one. Some examples of data augmentation techniques are cropping, rotating, changing colors, flipping, among others. This method is used to augment the data and to present the model with new perspectives of the same image; with this, the model is able to capture relevant information from the data, regardless of its variations.

Another component of CL is a **network of encoders**. A network of this type takes augmented instances as input and maps them to a latent representation space, where features and similarities are identified. By doing this, it facilitates the process of discriminating between similar and non-similar data in the following steps.

In order to perfect the learned representations, a **projection network** is used. This network takes as input the output of the encoder network, and projects it into a space with a smaller dimension, called **embedding space**. This step helps to increase the discriminative power of the learned representations, as it reduces the complexity and redundancy of the data, thus facilitating the process of distinguishing the instances.

The next step is the so called **contrastive learning** . This is applied when the augmented instances are encoded in the same embedding space. The aim is then to maximize the compatibility of positive data while distancing negative pairs. Usually, similarity is measured using a distance metric, such as cosine similarity or Euclidean distance.

As with all machine learning models, a **loss function** is required. There are several loss functions that are used in contrastive methods, and these have a crucial role of leading the model to capture meaningful representations and differentiate between similar and dissimilar instances. The choice of the loss function depends of the requirements of the task to be

developed. The next section takes a closer look at the most widely used loss functions, as these have contributed greatly to the development of contrastive learning methods.
Once the loss function has been defined, the model can be trained. The goal is to minimize the loss function. Optimization algorithms such as Stochastic Gradient Descent (SGD) are usually used to fine-tune the parameters of the model . The rest of the training takes place normally - the model learns to capture the relevant features in the data, and as the training progresses, the model is expected to be improving, which leads to better discrimination between the data at the end of the training.
Evaluation and generalization are very important steps in assessing the quality of the learned representations and their efficiency in practice. One way to evaluate the performance of the model is by observing its performance on downstream tasks. The learned representations will be used for specific tasks, and performance is then assessed using various metrics, such as accuracy, precision, F1 score, etc. Another way of evaluating is through transfer learning, i.e., applying what has been learned to a related task. Better performance indicates better generalization and usefulness of the learned representations in both cases.

2.3 Related Work

This section reviews the main work done in the area of Contrastive Learning.
The diversity of proposed methodologies is systematically arranged and presented, categorized based on the nature of their contributions, be it in the realm of loss functions, architectural innovations, or any other aspects.

2.3.1 Contrastive Loss

The initial idea of contrastive learning was to compare a pair of images in its loss function, called Contrastive Loss. But various methods have been built up from the initial idea, where more examples (both negative and positive) are used.

2.3.2 Triplet Loss

The concept of **Triplet Loss**[8] was initially introduced to solve the problem of learning fine grained image similarity. The authors proposed a technique that learns the similarity of images directly from them (not from their features), and for this they proposed Triplet Loss - for each training sample, a triplet of data is required, in which, given an **anchor input**, a **positive sample** (belonging to the same class as the anchor) and a **negative sample**

(sampled from another class) are selected. The idea is to minimize the distance between the anchor and the positive sample and to maximize the distance between the anchor and the negative example. This is visualized in the Figure 2.1. They showed that it is more efficient to tackle the problem this way than in previous ways (for example, first extracting the features and then learning the image similarity models based on these features) and a crucial factor for this success is the choice of a good trio of images.

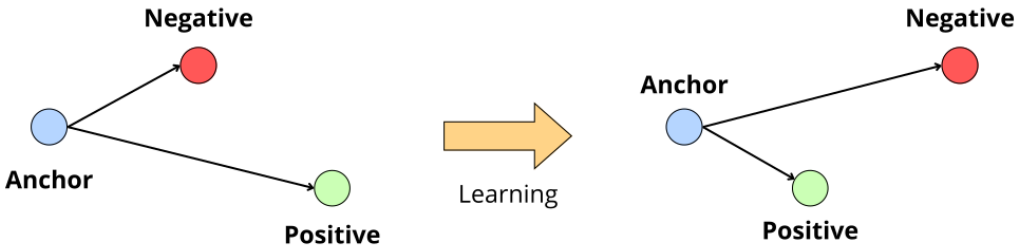

Figure 2.1: Illustration of Triplet Loss given one positive and one negative example per anchor.

The loss function of this method is given by the Equation 2.1:

$$l(p_i, p_i^+, p_i^-) = \max\left\{0, g + D\left(f(p_i), f(p_i^+)\right) - D\left(f(p_i), f(p_i^-)\right)\right\}, \quad \textbf{(2.1)}$$

where (p_i, p_i^+, p_i^-) are the query image, positive image and negative image, respectively, $f(\cdot)$ is the learned embedding function, $D(a, b) = \|a - b\|^2$ is the squared Euclidean distance and g is a margin parameter that enforces a minimum distance gap between the anchor-positive pair and the anchor-negative pair.

Other Works using Triplet Loss

Several authors have based their research on the idea of triplet loss. Some examples of this are shown below.

Based on the idea of Triplet Loss, [17] tackles this method but in videos. The idea is that two segments of a video connected with a "track" have similar representations since they may belong to the same object. Given an unlabeled video, unsupervised tracking is performed on its frames. Trios of theses frames are then formed, which include the initial frame (*query*), a tracked frame in the last frame (positive example) and a random frame from another video (negative example). All this is given to a siamese triplet network [1].

[1] A Siamese Neural Network is a class of neural network architectures that, in this case, contains three identical sub-networks, e.g., the networks have the same configurations, the same parameters and the same weights.

[18] defends this type of loss function for Person Re-Identification (ReID). They use a simple Convolutional Neural Network (CNN) with triplet loss to prove that this loss function is good in cases of person ReID. They propose a modification to the triplet loss function, called Batch Hard: the idea is to form several batches of PK elements, where P classes are randomly sampled (e.g. identity of people) and K elements from each one of these classes are also sampled (e.g. people); then, for each element in a batch, they find its "most positive" sample (hard positive) and its "most negative" sample (hard negative), so that trios are formed to calculate its loss.

[19] uses three neural networks to learn representations based on distance comparisons. The authors found that representations were less neglected when Siamese networks were used. The problem is that these networks are sensitive to the classification of what is or is not similar. They then used three networks, with shared parameters. The inputs were three examples, a base example (*query*), one from the same class (positive) and one from another class (negative) and the output was two numbers representing the distances (between the *query* and the other examples). Using a simple loss function, the team was able to obtain good results.

The concept of triplet loss, a pivotal framework in the realm of contrastive learning, has undergone extensive elaboration and refinement in numerous articles. These publications ([9], [10]) delve into the intricacies of the triplet loss methodology, pushing the boundaries by incorporating a more comprehensive array of negative examples for a nuanced and in-depth exploration of its applications and potential enhancements.

2.3.3 Lifted Structured Loss

One of the methods that incorporated more negative examples was Lifted Structured Loss [9]. The **Lifted Structured Loss** utilizes **all the pairwise edges** within one training batch. Until then, the models were trained so that the network simultaneously learned a representation of the features and embeddings with semantic meaning. The strategies proposed so far did not take full advantage of the training batches, and the Lifted Structured Loss transforms the vector of distances between pairs within the same batch into the distance matrix between pairs. To do this, a new loss function was proposed in which, if, for example, a batch had six elements, all these elements would be compared, and not separated into trios and pairs. This can be seen in the Figure 2.2, where a comparison is made between

the contrastive, triplet and lifted structured embeddings. Something important to note is that the elements in the batches are not random - the negative examples are selected using a technique called hard negative mining.

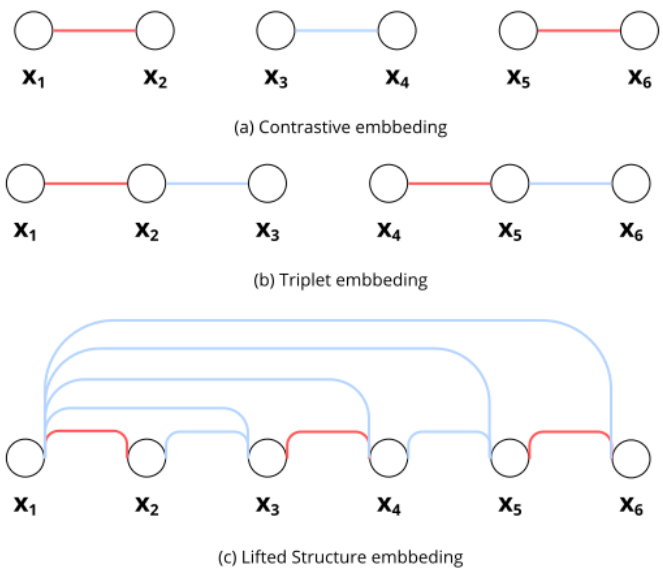

Figure 2.2: An illustrative comparison is made among Contrastive Loss, Triplet Loss, and Lifted Structured Loss. Similar sample pairs are denoted by red edges, while dissimilar pairs are represented by blue edges. Note that not all of the edges are shown for a better visualization of the figure.

The loss function, per batch, of the lifted structured method is given by the equations 2.2 and 2.3:

$$\tilde{J}_{i,j} = \log \left(\sum_{(i,k)\in\mathcal{N}} \exp\{\alpha - D_{i,k}\} + \sum_{(j,l)\in\mathcal{N}} \exp\{\alpha - D_{j,l}\} \right) + D_{i,j}, \quad (2.2)$$

$$\tilde{J} = \frac{1}{2|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \max\left(0,\, \tilde{J}_{i,j}\right)^2, \quad (2.3)$$

where $\tilde{J}_{i,j}$ measures the similarity between pairs (i, j), $\mathcal{N}$ and $\mathcal{P}$ are the sets of negative and positive pairs, respectively, $D_{i,j}$ gives the distance between samples and α is the margin parameter that defines the separation boundary between positive and negative pairs. The objective is to minimize $\tilde{J}$ to pull together positive pairs and push apart negative pairs.

2.3.4 N-pair Loss

N-pair Loss is a loss function that was proposed in [10], because the authors stated that, although contrastive loss and triplet loss are good, they suffer from slow convergence, and the cause of which is the use of only one negative example (there is no interaction between other elements of negative classes in each update). So, the new proposed loss function here is a generalization of the triplet loss - the idea is to have an $(N+1)$ set of elements, in which there is a *query* example, a positive example and $N-1$ **negative examples**. This is called N-pair Loss. The idea here is that all the negative elements are "pushed" away from the *query* example at the same time, according to their similarity to it. There are other details mentioned in the article, such as the definition of the loss function, which is done in such a way as to be computationally efficient, and also the selection of the negative examples. Figure 2.3 illustrates how this method works.

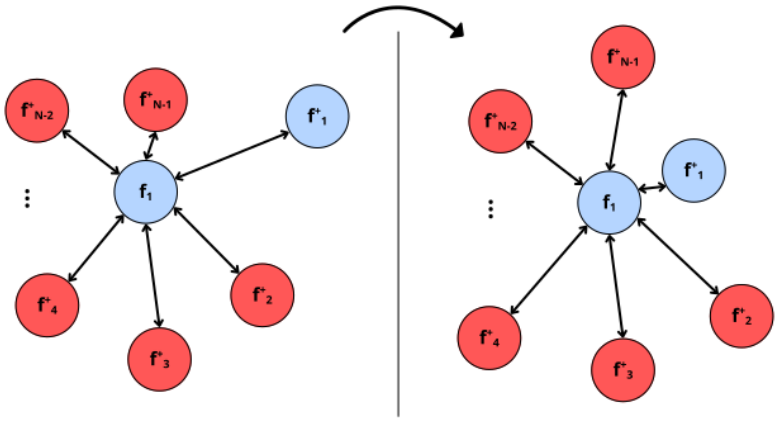

Figure 2.3: Illustration of N-pair Loss, where the $(N+1)$-tuplet loss pushes $N-1$ negative examples all at once, based on their similarity to the anchor example.

The loss function of this approach is given by the equation 2.4:

$$\mathcal{L}_{\text{N-pair}} = \log\left(1 + \sum_{i=1}^{N-1} \exp\left(f^\top f_i - f^\top f^+\right)\right), \quad (2.4)$$

where f is the embedding of the anchor sample, f^+ is the embedding of the positive sample, f_i are the embeddings of the negative samples, $\top$ denotes the transpose and N is the number of classes or distinct samples in the batch.

2.3.5 Noise Contrastive Estimation

So far, the main contributions made in the field of CL are closely linked to the loss function, especially when it comes to adding negative examples to it. However, the method described next does not tackle the problem in this way. This next technique is known as **Noise Contrastive Estimation (NCE)** [20]. The main idea behind NCE is to turn the CL problem from a problem of estimating a complex probabilistic model into a simpler classification problem. Instead of directly estimating the probabilities of observing the data, this method introduces artificially generated noise (negative samples) and frames the problem as distinguishing between real points and noisy samples (performs a non-linear logistic regression between them). This method is able to consistently estimate sophisticated statistical models that do not need to be normalized (the normalization constant is estimated like any other parameter in the model). Also, this mechanism is computationally efficient. Overall, this procedure greatly revolutionized how the loss function was tackled, and showed a new connection between supervised and unsupervised learning. Originally, the loss function receives only a positive example and a sample with noise - but this function is widely used and there have already been several proposals for models with alterations to this function, as it will be shown ahead.

Other Works using Noise Contrastive Estimation

The idea behind [21] is to use each example as if it were a class - that is, the number of classes is the size of the dataset used. This is an unsupervised method, and the idea is that the model learns the similarities of the instances without any "class" annotation - if the features are similar it is assumed that they are either the same object or a similar object.

2.3.6 InfoMax Noise Contrastive Estimation

In the work [22], a new loss function is built from NCE. This is an **unsupervised learning** method, and its main idea is to learn representations by predicting the future in latent space (using autoregressive models). This is done by compressing high-dimensional information into a compact latent space, where predictions are easier to make; then an autoregressive model (in latent space) is used to try to predict future observations. NCE is used as the loss function, but modified - the encoder and the autoregressive model are trained together to optimize the loss based on the NCE, which they call **InfoMax Noise Contrastive Estimation (InfoNCE)**. Like NCE, this is loss function is widely used in the community.

Other Works using InfoNCE

One assumption when using CL is that positive pairs should share some information, since they are positive examples. What if this assumption were broken? What would happen if this did not happen? Some works suggests that CL representations are better in the presence of noisy views (false positive pairs). Based on this idea, [23] proposes a new loss function that does just that. To do this, the authors began by understanding the connections between CL and binary classification with noise in supervised environments. They then proposed a new loss function, called **Robust InfoNCE (RINCE)**, a contrastive loss function that satisfies the symmetric condition (for example, the quadratic function), which has been shown to be beneficial in these cases.

Many of the examples that have been seen have heavy computational configurations, whether it is the number of epochs, the size of the batch sample, etc. What the authors of [24] found is that this could be simplified by modifying a term in the much-used InfoNCE loss function. The authors noticed that the use of a term, identified by them as Negative-Positive-Coupling (NPC), reduces the effectiveness of the algorithm, and that the cause of this is the fact that easy SSL tasks are used. The authors did a lot of in-depth research into this aspect, checking the relationship between this term and the batch size in a well-known method, Simple Contrastive Learning Representation (SimCLR). By removing that term, there is a relief in this size. A loss function was then proposed, called **Decoupled Contrastive Learning (DCL)**, which is less sensitive to the parameters that the authors considered to be crucial when training contrastive models.

2.3.7 The Importance of Sampling and Structure

So far, the greatest contributions to the topic of CL have focused on the loss function and its importance. However, there are other factors of importance beyond just this.
In many of these methods the part of selecting the examples to use during training is neglected. [12] shows that **sampling matters**. In fact, this article demonstrates how important the selection of examples is, and even in simpler loss functions, such as contrastive loss or triplet loss, the results improve when the data selection is better. They also proposed a new loss function, but only as an extension of the contrastive loss, where the positive examples are encouraged to be at a certain distance from each other instead of being as close to each other as possible. Focusing now on the sampling part, the idea is to sample uniformly according to distance (sampling with weights), which gives examples that are more spread

out rather than all clustered around a small region. In Figure 2.4, it is possible to see an illustration of this strategy.

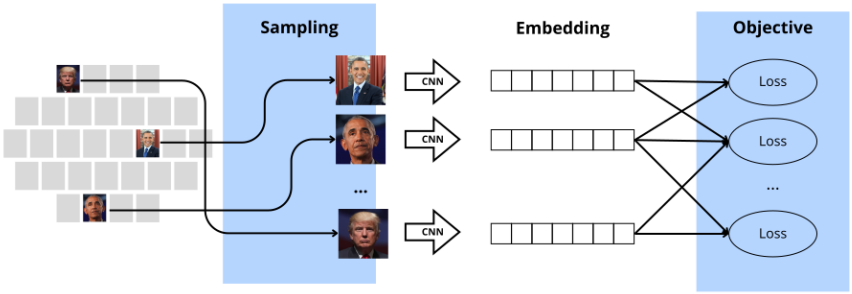

Figure 2.4: Illustration of Sampling Matters. In the first step, images are sampled and grouped into a batch, then a deep network transforms them into embeddings. Lastly, a loss function evaluates the embedding quality, with both sampling and the loss function affecting the overall training goal.

In addition to the importance of the loss function and sampling in Contrastive Learning, [25] shows that **structure** is also important. This method, **Deep InfoMax (DIM)**, is based on the simple idea of training an encoder to maximize the mutual information between its input and output.

2.3.8 Momentum Contrast

Momentum Contrast (MoCo) [13] is a **self-supervised** contrastive learning framework, known for its approach to building large, consistent dictionaries in an unsupervised manner through contrastive loss. The primary focus of MoCo is on the creation of a dynamic dictionary of negative instances, a concept pivotal to the ability of the model to discern meaningful features and similarities within the data.

In the realm of loss functions, MoCo utilizes the InfoNCE contrastive loss function. This function serves as an unsupervised objective to train a network of encoders responsible for representing queries and keys. The distinctive feature of MoCo lies in its use of two encoder networks: an online encoder processing current data and a target encoder, which embodies a moving average of the online encoder parameters.

The key innovation introduced by MoCo is the moment update for the target encoder. Unlike methods relying on fixed dictionaries for negative samples, the moving average target encoder of MoCo offers a more stable and effective source of negatives for contrastive loss. This strategic choice not only enhances the ability of the model to capture relevant information but also contributes to computational efficiency. By dynamically updating the dictionary of negative instances, MoCo ensures a richer set of contrasting examples for learning representations. This is visualized in Figure 2.5. This dynamic dictionary concept propels the

performance of MoCo across diverse domains, including computer vision and natural language processing. The framework excels in maximizing agreement between positive pairs while minimizing agreement between negative pairs. MoCo has demonstrated its prowess by achieving state-of-the-art results in multiple benchmark datasets, solidifying its position as a leading methodology in the field of self-supervised contrastive learning.

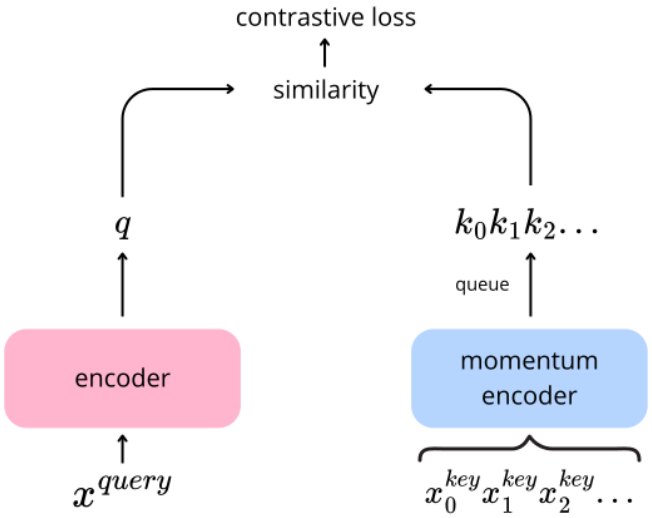

Figure 2.5: Illustration of how Momentum Contrast learns visual representations.

The loss function of the MoCo method is given by the Equation 2.5:

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)}, \quad (2.5)$$

where q is the query representation (the current encoded sample), k_+ is the positive key (matching sample from the memory queue), k_i are the negative keys stored in the dictionary queue, τ is the temperature parameter and K is the total number of keys in the dictionary.

2.3.9 Simple Contrastive Learning Representation

The parallel augmentations category produces two noisy versions of an anchor image and here the goal becomes to learn a representation such that these two altered versions share the same embedding. This is where the **SimCLR** [14] is. SimCLR is a framework for **unsupervised** learning, specifically designed for computer vision tasks. The method focuses on the principle of maximizing the agreement between augmented views of the same instance while simultaneously minimizing agreement between views from different instances. This approach ensures that SimCLR captures meaningful similarities and differences within the

data, contributing to the learning of robust and versatile representations.

At the heart of the methodology of SimCLR lies a Siamese network architecture, featuring two identical Neural Network (NN) or encoders that share the same weights. Each encoder processes one of the augmented views and produces a feature representation. The choice of three specific transformations — random cropping, random color distortions, and random Gaussian blur — emphasizes the critical role of data augmentation in defining effective prediction tasks. The design of SimCLR also incorporates a non-linear transformation between representations and employs a loss function known as Normalized Temperature-scaled cross-entropy (NT-Xent). This loss function encourages positive pairs to have similar representations while pushing away negative pairs, ultimately enhancing the quality of the learned representations.

The evaluation of SimCLR involves a linear evaluation protocol, where a linear classifier is added atop the frozen encoder and trained on a small labeled dataset. The performance of this classifier serves as a metric for assessing the quality of the learned features.

In Figure 2.6 it is possible to see the framework of SimCLR as well as its major components.

The versatility of this method extends beyond computer vision to encompass domains such as natural language processing and reinforcement learning. Its remarkable performance across various benchmarks and tasks underscores its capacity to learn powerful representations without relying on explicit labels. The combination of data augmentation, a well-crafted contrastive objective, and a symmetric neural network architecture solidifies status of SimCLR as a leading framework in the realm of self-supervised contrastive learning.

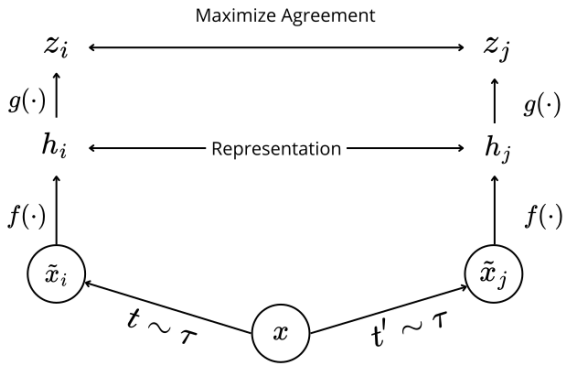

Figure 2.6: A simple framework for contrastive learning of visual representations.

The loss function of this method is given by the following Equation (2.6):

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)}, \quad (2.6)$$

where z_i and z_j are the embeddings of two augmented views of the same image, $\text{sim}(z_i, z_j)$ is the cosine similarity between the embeddings, τ is the temperature parameter and $\mathbb{I}_{[k \neq i]}$ is an indicator function (1 when $k \neq i$, 0 otherwise).

2.3.10 Bootstrap Your Own Latent

Unlike all the previous methods, **Bootstrap Your Own Latent (BYOL)** [15] is a pioneering **unsupervised** learning approach **that challenges the necessity of using negative examples**. BYOL utilizes a dual-network structure comprising online and target networks, both featuring an encoder, a projector, and a predictor with distinct weights. The key innovation lies in the online network training to predict the representation of the target network of a differently augmented view of the same image. This process promotes stability as the target network evolves gradually.

In the context of self-supervised contrastive learning, BYOL emphasizes continuous updates to the target network parameters through exponential moving averages of the weights of the online network. This unique approach is designed to maximize agreement between augmented views of identical instances, effectively decoupling the similarity estimation from the traditional use of negative examples.

The shared architecture between the online and target networks, coupled with the incorporation of a predictor, plays a crucial role in preventing network collapse. The predictor serves as a strategic component, contributing to the overall robustness of the framework. This can be seen in Figure 2.7.

BYOL has garnered considerable attention for its remarkable performance across diverse domains, including computer vision and natural language processing. Notably, it has achieved state-of-the-art results in multiple benchmark datasets, showcasing its prowess in enhancing representation quality and pushing the boundaries of unsupervised learning methodologies.

The loss function of this method can be observed in Equation 2.7:

$$\mathcal{L}_{\theta,\xi} \triangleq \left\| \overline{q_\theta}(z_\theta) - \overline{z}'_\xi \right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \left\| z'_\xi \right\|_2}, \quad (2.7)$$

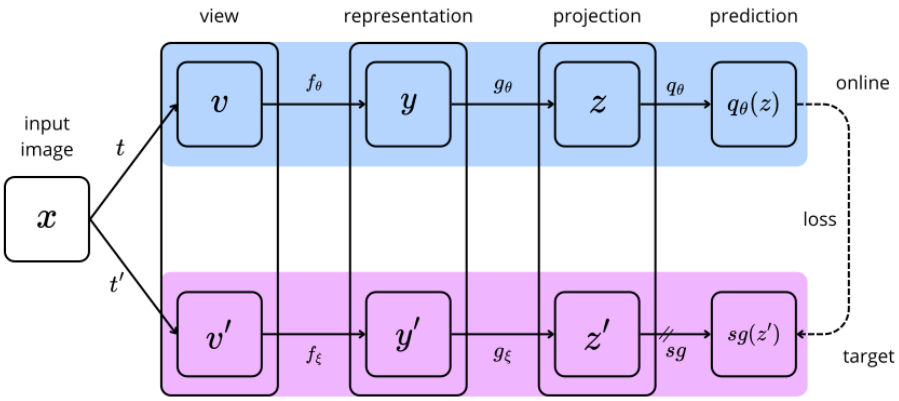

Figure 2.7: The model architecture of BYOL.

where θ is the parameter of the online network (updated by gradient descent), ξ is the parameter of the target network (updated via exponential moving average of θ), z_θ is the representation from the projector of the online network, z'_ξ is the representation from the projector of the target network and q_θ is the predictor head of the online network.

2.3.11 Swapping Assignments between multiple Views

The algorithm introduced in [26], known as **Swapping Assignments between multiple Views of the same image (SwAV)**, stands out as a novel online **unsupervised** learning approach. Unlike traditional contrastive learning methods, SwAV, or Swapped Augmentations and Views, adopts a distinctive strategy that deviates from pairwise image comparisons. Instead of directly assessing whether two images belong to the same class or initial image, SwAV operates by grouping data into clusters. The algorithm calculates representations online, ensuring coherence across different views of the same image within these clusters. This approach enables the contrast of various image views without relying on explicit pairwise feature comparisons.

At the core of the methodology of SwAV is the utilization of clustering-based objectives, leveraging multiple augmentations of the same image and diverse views within a mini-batch. This encourages the model to assign similar representations to augmented views of the same instance. The authors of SwAV also propose an innovative data augmentation technique called multi-crop, introducing a mixture of views with different resolutions to enhance the learning process. Figure 2.8 provides the architecture of SwAV.

The efficiency of SwAV extends beyond its unique approach, as it attains results in diverse computer vision tasks such as image classification and object detection. Thanks to its clustering-based objectives, SwAV excels in identifying clusters of similar representations without ex-

plicit class labels. This has led to competitive performance on various benchmark datasets, highlighting the potential of SwAV for advancing the field of unsupervised contrastive learning.

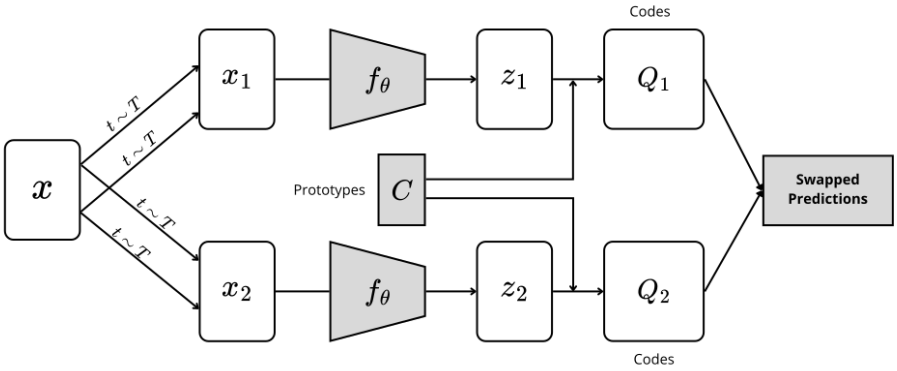

Figure 2.8: The model architecture of SwAV.

The loss function of this method is given by Equation 2.8:

$$\ell(\mathbf{z}_t, \mathbf{q}_s) = -\sum_{k=1}^{K} \mathbf{q}_s^{(k)} \log \mathbf{p}_t^{(k)} \quad (2.8)$$

where the softmax probability $\mathbf{p}_t^{(k)}$ is computed as it can be seen in Equation 2.9:

$$\mathbf{p}_t^{(k)} = \frac{\exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_k\right)}{\sum_{k'=1}^{K} \exp\left(\frac{1}{\tau}\mathbf{z}_t^\top \mathbf{c}_{k'}\right)} \quad (2.9)$$

where $\mathbf{z}_t$ is the feature representation from the target view, $\mathbf{q}_s$ is the code (pseudo-label) from the source view assignment, $\mathbf{c}_k$ is the k-th prototype vector, K is the number of prototypes and τ is the temperature parameter.

2.3.12 Barlow Twins

Based on the foundational work of neuroscientist H. Barlow, the next method proposed is **Barlow Twins** [27], a **self-supervised contrastive learning framework**. This innovative approach suggests a function designed to prevent collapse by assessing the cross-correlation matrix between the outputs of two identical networks. These networks receive distorted versions of a given sample, and the objective is to minimize the deviation of the cross-correlation matrix from the identity matrix. By doing so, the embedding vectors of the altered versions become more similar, effectively reducing redundancy between their

components. The method is rooted in unsupervised learning and employs the concept of redundancy reduction.

Barlow Twins is centered around diminishing cross-correlation between latent representations. This framework introduces a decorrelation loss, which encourages the model to generate diverse representations for similar instances, thereby enhancing overall discriminative power. The primary goal of Barlow Twins is to decrease cross-correlation, enabling the capture of more unique and informative features in the learned representations. Demonstrating its effectiveness across various domains, including computer vision and natural language processing, Barlow Twins has achieved state-of-the-art results in multiple benchmark datasets, showcasing its prowess and versatility. Figure 2.9 provides an illustration of Barlow Twins learning pipeline.

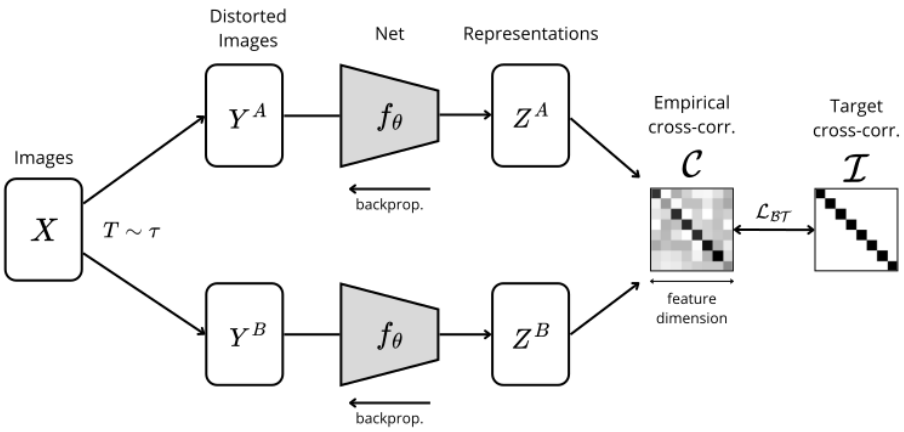

Figure 2.9: Illustration of Barlow Twins learning pipeline.

The following Equation 2.10 gives the loss function for this method:

$$\mathcal{L}_{BT} = \underbrace{\sum_i (1 - C_{ii})^2}_{\text{Invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} C_{ij}^2}_{\text{Redundancy reduction term}}, \quad (2.10)$$

where C is the cross-correlation matrix computed between two identical networks fed with different augmented views of the same batch, C_{ii} are the diagonal elements of C, C_{ij} are the off-diagonal elements of C and λ is a positive constant trading off the importance of the first and second terms of the loss.

2.3.13 Supervised Contrastive Learning

Some of the work so far focuses heavily on adding negative examples to the loss function. But this raises a question - why not also use more than one positive example? This is the main idea behind the article [11], which proposes a loss function called **SupCon** . This is a **supervised** learning method that tries to get the most out of labels, and the idea here is to use more positive examples. Here, these examples are taken from the same class as the anchor example, rather than being examples of data augmentation of the base example, as in unsupervised methods, which will be talked about later on.

Figure 2.10 shows the proposed method in comparison with self-supervised methods.

The loss function they propose can be seen as a generalization of the triplet loss or the N-pair loss.

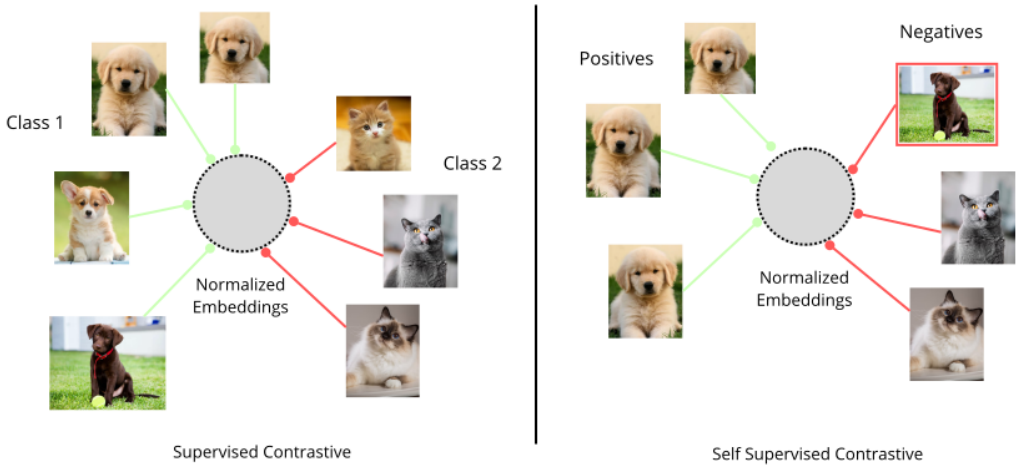

Figure 2.10: Self-supervised vs. SupCon.

The loss function of this method is given by Equation 2.11:

$$\mathcal{L}_{out}^{sup} = \sum_{i \in I} \mathcal{L}_{out,i}^{sup} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(z_i \cdot z_p / \tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a / \tau)}, \quad (2.11)$$

where I is the set of all samples in the batch, $P(i)$ is the set of positives for the anchor i, $A(i)$ is the set of all samples except i, z_i is the normalized embedding of the anchor sample, z_p is the normalized embedding of the positive sample p, z_a is the normalized embedding of any sample (positive or negative) and τ e the temperature parameter.

The $\mathcal{L}_{out}^{sup}$ means that the summation over positives is located *outside* of the log. The authors also wrote the equivalent of this function but with the summation *inside* if the log, but both functions are equivalent and therefore only one is shown here.

Since the topic of supervised learning is being discussed, mentioning the article [28] is pertinent, especially since it makes a comparison with the previous method, SupCon.
The article mentioned just above describes the fact that data is often unbalanced, i.e. the long-tailed distribution problem - there are some classes that contain more examples, while most classes have few instances. The problem with learning in unbalanced settings is that the classes with the lowest frequency tend to be overlapped by the classes with the highest frequency. To solve this problem, the authors suggest **Parametric Contrastive Learning (PaCo)**, a set of parametric centers learnable by class in **supervised CL**. This proves that the optimum values for the probability of two samples being a true positive pair, varying from the most frequent class to the least frequent class, are more balanced. In other words, this means that the model handles classes with fewer instances more carefully, making PaCo beneficial for this type of learning. PaCo has several components, such as a data augmentation strategy, a loss function, a momentum encoder and the size of the queue. However, after analyzing these components, the authors found that the performance of the method was boosted when the momentum encoder was removed, giving rise to **Generalized Parametric Contrastive Learning (GPaCo)**.

2.3.14 Self-Contrastive Learning

Many Contrastive Learning methods use a multi-viewed framework, which takes advantage of two important topics, label information and data augmentation. In fact, the SupCon method has this structure. A framework of this type proves to be crucial, since if a single-viewed framework is used (i.e. where only label information is used), performance is significantly reduced. However, the multi-viewed framework makes training time and memory usage very expensive. In order to implement a multi-viewed framework that does not use data augmentation, the **SelfCon** [29] method was proposed. This consists of using a multi-exit architecture with sub-networks that produce multiple features from a single image. By having an architecture of this type, SelfCon self-contrasts with the multiple outputs from the different levels of a single network, which means that a single-viewed framework can be used successfully.
Comparing the SupCon and SelfCon methods, the former relies on augmentations-based multi-views, while the latter is a single-viewed Supervised Contrastive Learning framework, where several features of the same image are produced using a sub-network. A comparison of these two methods can be seen in Figure 2.11.

The sub-network in this methodology plays the role of the augmentations and gives an alternative view in the feature space. It should be noted that the authors of this article, in addition to implementing their SelfCon method, also implemented SupCon, SupCon with a single-viewed batch and SelfCon with a multi-viewed batch, so that they could observe the effects of the single-viewed framework and also of the sub-network. This method proved to be very promising, obtaining results as good as or better than other existing methods.

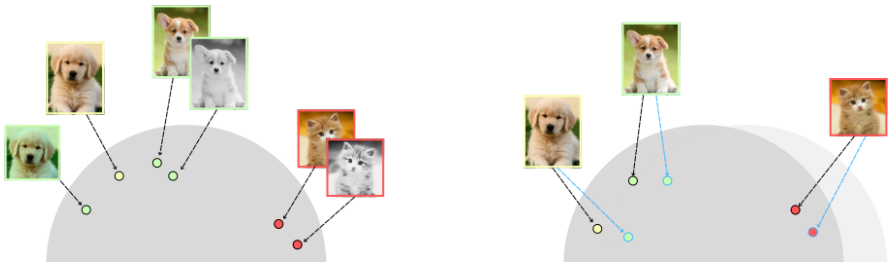

Figure 2.11: Overview of (a) SupCon learning and (b) SelfCon learning.

The loss function of the SelfCon method is given by Equation 2.12:

$$L_{self} = \sum_{\substack{i \in I \\ \omega \in \Omega}} \left[-\frac{1}{|P_{i1}||\Omega|} \sum_{\substack{p_1 \in P_{i1} \\ \omega_1 \in \Omega}} \omega(x_i)^\top \omega_1(x_{p_1}) + \log \sum_{\omega_2 \in \Omega} \left(\sum_{p_2 \in P_{i2}} e^{\omega(x_i)^\top \omega_2(x_{p_2})} + \sum_{n \in N_i} e^{\omega(x_i)^\top \omega_2(x_n)} \right) \right], \quad (2.12)$$

where $\Omega = \{F, G\}$ is a function set of the backbone network (F) and the sub-network (G), P_{i1}, P_{i2} are sets of positive samples for anchor i and N_i is the set of negative samples for anchor i. In other words, the anchor is marked with the letter i, positive pairs are marked with the letter p and negative pairs are marked with the letter n.

2.3.15 Unsupervised Methods

Although many of the methods presented were already unsupervised learning, for many years it was supervised learning that dominated this topic. Supervised learning methods have had the best results when it comes to computer vision. However, unsupervised learning methods are closing the gap on the performance of supervised methods. Recent self-supervised learning methods are based on a combination of two factors: a contrastive loss function and a set of data augmentations. Many of the methods to follow focus on improving these aspects in order to achieve better results and for unsupervised methods to become increasingly used in

the field of computer vision.

The following method is an example of an unsupervised learning use case, but in this case in the field of robotics. The article [30] explores the learning of robot representations and behaviors from unlabeled videos, focusing on using contrastive learning to discern similarities and differences in images/videos. Motivated by the challenges of supervising robots, the goal is for robots to autonomously grasp relevant attributes of object interactions and human "poses" through observation. The algorithm builds on existing ones, aiming for embeddings sensitive to object interactions and poses while remaining insensitive to nuisance variations. Triplet loss, emphasizing the proximity of anchor and positive frames versus a distant negative frame, is a key focus. During training, the **Time Contrastive Network (TCN)** addresses questions about learning indicative visual representations, integrating with reinforcement learning for complex object manipulation, and enabling real-time imitation of human poses without explicit joint correspondences. Encouragingly, this method shows promising results in the field of robotics.

In the work [31], there is a focus on state representation learning, a vital aspect for intelligent agent development, aiming to capture generative latent factors by maximizing temporal and spatial mutual information. The motivation stems from the human ability to perceive and describe the world without supervision, unlike previous supervised or problematic unsupervised computational methods. The proposed approach, **Spatiotemporal Deep InfoMax (ST-DIM)**, employs contrastive learning with positive and negative examples. Two loss functions, global-local and local-local objectives, maximize mutual information across total observations and local features at different times. The InfoNCE score function, coupled with a bilinear model, enhances the contrastive state representation learning method.

The authors of the article [32] propose a self-learning method that maximizes the mutual information between features that are extracted from different views of the same task. The essential component here is that, by learning these features from different points of view, the model learns which are the most important characteristics of what it is learning (what has the most influence). These different viewpoints can be different angles, or even different types of data (image and sound, e.g.). The method they use is to utilize an image and the different examples of this image are different data augmentation methods used on that image. This method is called **Augmented Multiscale DIM (AMDIM)**, and is an extension of the DIM method. The main differences are that this method uses a more powerful encoder, predicts features through independent augmented versions of each input and predicts features simultaneously at several scales. NCE is also used here as a loss function.

As mentioned earlier, the idea behind most unsupervised learning methods is based on instance discrimination. And for this, you need to use data augmentation methods, but they come at a cost - they only alter the image superficially. So the authors of [33] suggest a way of altering images in a natural way - videos. They propose **Video Noise Contrastive Estimation (VINCE)**, a method that combines the use of unlabeled videos with NCE. The authors argue that videos have more significant semantic information and that, when using videos, they also have temporal information. To do this, they train a network so that it can identify whether or not two images belong to the same video, which leads the network to try to understand how objects change. They also generalize NCE to use several positive examples, with the help of MoCo. Overall, they have obtained very positive results, and the authors believe that this will be the future of computer vision.

Also focused on videos, [34] introduces us to **Video Deep InfoMax (VDIM)**, an extension of DIM to the video domain, where its structure is altered so that spatial and temporal components are taken into account. The authors observed that the visualization of frames from natural sequences (videos) with a reduced sampling interval between these frames achieved good results. Again, the premise of this article is that images are too static, they do not have as much information about objects or their nature as videos. How does one expect the result of a network to be good and for it to understand the interaction of objects if what is given to the network to train is not good enough? The authors use DIM because they realize that they can modify it to understand the role of different points of view of the same scene but in a space-time setting. The proposed model has components from a previous example, AMDIM. The authors show that not only is this space-time setting a strong promise for the self supervised learning domain, but they also show that data augmentation in videos plays an important role for unsupervised learning methods in general.

The method proposed in [35], called **Local Aggregation (LA)** is a procedure that trains an embedding function to maximize an allocation metric so that similar data instances are together in the embedding space, and dissimilar ones are further apart. They try to find a balance between this distance (not too close and not too far). This is an unsupervised learning method. By simultaneously optimizing this smooth clustering structure and the non-linear embedding in which it is performed, this method exposes statistical regularities in the data.

[36] presents an unsupervised learning method called **Prototypical Contrastive Learning (PCL)**. This methodology links contrastive learning and clustering. In addition to learning low-level features for discriminating examples, PCL encodes semantic patterns found by clustering in the learned embedding space, which is of greater importance. In addition to this

proposed method, the authors also propose a new loss function, called ProtoNCE, a loss function based on InfoNCE, but more generalized, which promotes representations that resemble their corresponding prototypes (each instance has several prototypes and the contrastive loss function created forces the embedding of a sample to be closer to related prototypes than to others). One of the problems the authors pointed out with previous methods was that the representations created did not encode the semantic structure of the data, and that negative examples were treated as such because the semantic part was not taken into account.

Self-supervised contrastive methods often consider certain invariances, and these methods failed when something violate this restriction. The idea behind the article [37] is to capture variation and invariance in images, and the key behind this method is the use of separate embedding spaces, where each of these spaces is invariant to all but one variation (augmentation). The name given to this method is **Leave-one-out Contrastive Learning (LooC)**. The augmentations considered in this method are rotation, color jittering and texture randomization. Its base framework is MoCo and InfoNCE is used as the loss function.

When alterations are made to images using data augmentation methods, they are usually "weaker" alterations, so that the identity of the image is still preserved and useful representations can be drawn from the altered images. But this limits the exploration of other patterns in the images, which can be beneficial. The use of "strong" alterations alone is also not indicated, because it makes it difficult to collect the features. Therefore, a distribution between weakly and strongly augmented images in the representation database was proposed in the article [38], to help supervise the retrieval of strongly augmented queries from a set of instances. This method, called **Contrastive Learning with Stronger Augmentations (CLSA)**, can be used, according to the authors, in existing methods to improve their results. CLSA also proposes a set of data augmentation techniques to use, called "Stronger Augmentation", which combines 14 methods (including Equalize, Posterize, ShearX/Y, etc.).

2.4 Comparison between the main methods

This section presents a concise comparison of the main Contrastive Learning methods, with the aim of making it easier to analyze their fundamental characteristics and differences.
Table 2.1 summarizes the type of learning adopted by each method and the respective loss functions. Table 2.2 presents the datasets used and highlights the main strengths and limi-

tations of each approach, as well as other relevant information.

Note that the expressions of the loss functions are not explained in detail here, as including all the parameters would make the table too dense. For a complete and in-depth description of these functions, please consult the corresponding sections referenced in the Table 2.1 .

Table 2.1: Comparison of Contrastive Learning methods including the section where the method is described in more detail, the type of learning, and loss function.

Method	Section	Type of Learning	Loss Function
MoCo	2.3.8	Self-Supervised	$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)}$
SimCLR	2.3.9	Self-Supervised	$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)}$
BYOL	2.3.10	Self-Supervised	$\mathcal{L}_{\theta,\xi} = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\|q_\theta(z_\theta)\|_2 \cdot \|z'_\xi\|_2}$
SwAV	2.3.11	Self-Supervised	$\ell(\mathbf{z}_t, \mathbf{q}_s) = -\sum_{k=1}^{K} \mathbf{q}_s^{(k)} \log \mathbf{p}_t^{(k)}$
Barlow Twins	2.3.12	Self-Supervised	$\mathcal{L}_{BT} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2$
SupCon	2.3.13	Supervised	$\mathcal{L}_{out}^{sup} = \sum_{i \in I} \frac{-1}{\lvert P(i) \rvert} \sum_{p \in P(i)} \log \frac{\exp(z_i \cdot z_p/\tau)}{\sum_{a \in A(i)} \exp(z_i \cdot z_a/\tau)}$
SelfCon	2.3.14	Supervised	$L_{\text{self}} = \sum_{\substack{i \in I \\ \omega \in \Omega}} \Big[-\frac{1}{\lvert P_{i1} \rvert \lvert \Omega \rvert} \sum_{\substack{p_1 \in P_{i1} \\ \omega_1 \in \Omega}} \omega(x_i)^\top \omega_1(x_{p_1}) + \log \sum_{\omega_2 \in \Omega} \Big(\sum_{p_2 \in P_{i2}} e^{\omega(x_i)^\top \omega_2(x_{p_2})} + \sum_{n \in N_i} e^{\omega(x_i)^\top \omega_2(x_n)} \Big) \Big]$

2.5 Conclusions

This chapter began by introducing the concept of Contrastive Learning, as well as its main constituents, so that all the concepts could be understood beforehand.

Next, a range of CL methods were presented. These methods focus on the various aspects that make up Contrastive Learning, from the loss function, something that is very much in focus in these works, but also the structure, data processing, data augmentation techniques, among others. And although all these methods are focused on computer vision, a division can also be made between methods that focus on identifying people, objects, video processing and even robotics.

The next section will discuss the methodology of this project, where the problem of this dissertation will be explored, as well as the solutions developed to mitigate this problem.

Table 2.2: Comparison of Contrastive Learning methods including the datasets used to train the method, their strengths and weaknesses and other important information.

Method	Datasets	Strengths	Weaknesses	Other Important Information
MoCo	ImageNet-1M, Instagram-1B	Momentum encoder and dynamic dictionary enhance representations and stability for strong downstream performance.	Large dictionary and contrastive learning increase compute/memory needs.	Queue-based dictionary and momentum encoder avoid large batches while maintaining embedding quality.
SimCLR	ImageNet ILSVRC-2012, CIFAR-10	Simple, scalable contrastive learning with strong transfer performance.	Demands large batches and high compute, limiting scalability.	Strong augmentations enable effective pre-training, beating complex methods.
BYOL	ImageNet ILSVRC-2012	Eliminates negative samples while maintaining strong downstream performance.	Strong but computationally heavy, with limited theoretical foundation.	Uses momentum network and strong augmentations for stable, negative-free learning.
SwAV	ImageNet	Avoids negatives via clustering, scales efficiently, and learns semantic representations.	Relies on heavy clustering, sensitive to initialization and hyperparameters.	Uses online clustering and strong augmentation for invariant features, outperforming other self-supervised methods.
Barlow Twins	ImageNet ILSVRC-2012	Reduces feature redundancy without negative pairs, providing efficient SOTA performance.	Requires careful tuning and its simpler architecture may limit representation power.	Achieves strong performance via redundancy reduction, avoiding large batches and negatives for compact, rich representations.
SupCon	ImageNet, CIFAR-10/100	Uses class labels to enhance representation learning, achieving state-of-the-art performance on downstream tasks.	Requires large batches and labeled data, limiting use in memory-constrained or unsupervised scenarios.	Beats self-supervised methods by leveraging class info for separation, even with scarce labels.
SelfCon	ImageNet-100, CIFAR-10/100, Tiny-ImageNet, ImageNet	Simplifies supervised contrastive learning with single-view training, reducing complexity and improving representation quality.	Relies on labeled data, offers limited augmentation flexibility, potentially reducing feature diversity.	Uses a sub-network for efficient single-view learning, reducing augmentation and computational costs.

Chapter 3

Methodology

3.1 Introduction

This chapter presents the adopted approach for solving the problem. In this chapter, the problem encountered in the study of Contrastive Learning methods is explained, followed by the proposed solutions to solve the problems encountered, where practical examples of each of the proposed solutions are also presented.

3.2 Problem

The main objective of this dissertation is an in-depth analysis of current Contrastive Learning methods, with a special focus on the task of visual object recognition. The purpose of this analysis is not only to understand the mechanisms underlying these methods, but also to identify their limitations and possible opportunities for improvement. The method selected for detailed study and implementation is SelfCon. SelfCon is a contrastive learning method that avoids costly data augmentation by using a multi-exit network. Unlike traditional multi-view frameworks like SupCon, which generate multiple views through augmentations, SelfCon extracts multiple features from different levels of a single network. These features act as alternative views, enabling effective contrastive learning in a single-view setting. This approach reduces memory and training time while achieving performance comparable to or better than augmentation-based methods, establishing it as the state-of-the-art technique for the base model.
During the detailed analysis of the SelfCon method, both from reading the article and inspecting the code provided by the authors, some gaps were identified:

- During the test phase, a close examination was made of how the predicted model produced the final classification predictions. Although the proposed mechanism had good results, it could be altered to take other examples into account and thus improve the final prediction;
- The dataset used to train and evaluate the model was CIFAR-100, which is organized into 100 classes, divided into 20 superclasses. When analyzing the distribution of sam-

ple across these superclasses, a relevant question arose: *was the model confusing classes belonging to the same superclass due to their semantic proximity?* This possibility raises questions about the robustness of the interclass separation promoted by the method, especially in scenarios with high semantic overlap.

In order to overcome the limitations identified, solutions were developed and explored that seek to mitigate the problems observed. The proposed solutions will be briefly presented below.

3.3 Proposed Solutions

This section explains the different solutions found to overcome the limitations. Several solutions emerged to try to address the first shortcoming found, all with one idea behind them - the k Nearest Neighbors (K-NN) algorithm. One of the proposed solutions aimed to improve the accuracy of the model by using the similarity between the features of the images, instead of just using the direct predictions of the classifier. A solution was introduced to consider the k most similar images based on the similarity between the feature vectors. The idea is that if several images with similar features are classified as the same class, it is likely that the *query* image also belongs to that class.

By using the similarity between feature vectors instead of relying solely on the direct prediction of the classifier, this solution takes advantage of the following concept: images that have similar feature vectors (i.e. that are close in the representation space) tend to belong to the same class. Furthermore, when the k most similar vectors (neighbors) are collected, the model uses a form of collective voting between similar examples. In this context, if there are several neighbors that are classified with the same class, the probability of a *query* image belonging to the same class increases.

The example below shows an application of this solution. For the eight element, `pred[7]`, its five predictions are as follows:

$$\texttt{pred}[7] = [24, 7, 45, 79, 6].$$

This means that element 7 could belong to class 24, or 7, and so on. Predictions are made in order of probability, i.e. element i is more likely to belong to class `pred[i][0]` than to class `pred[i][4]` [1].

[1] It is important to remember that we are dealing with the CIFAR-100 dataset, which has 100 different classes, here numbered between 0 and 99.

The five nearest neighbors of the eighth sample are

```
[166, 32, 106, 113, 84].
```

Each one of the neighbors of element 7 also has five predictions, and the set of these will be used to make the final prediction of the eighth element. This set of predictions is as follows:

```
[24, 7, 45, 79, 6, 24, 7, 6, 45, 25,
24, 14, 6, 7, 45, 24, 7, 45, 98, 86,
24, 32, 7, 77, 91, 79, 24, 6, 26, 7].
```

Figure 3.1 shows a visual example of what happens. In this case, the eighth element, $i = 7$ has as predictions the classes $24, 7, 45, 79$ and 6. After counting the classes of the element and its neighbors, the final prediction of this element is class 24, since it is the class that appears most often in the set of neighbors.

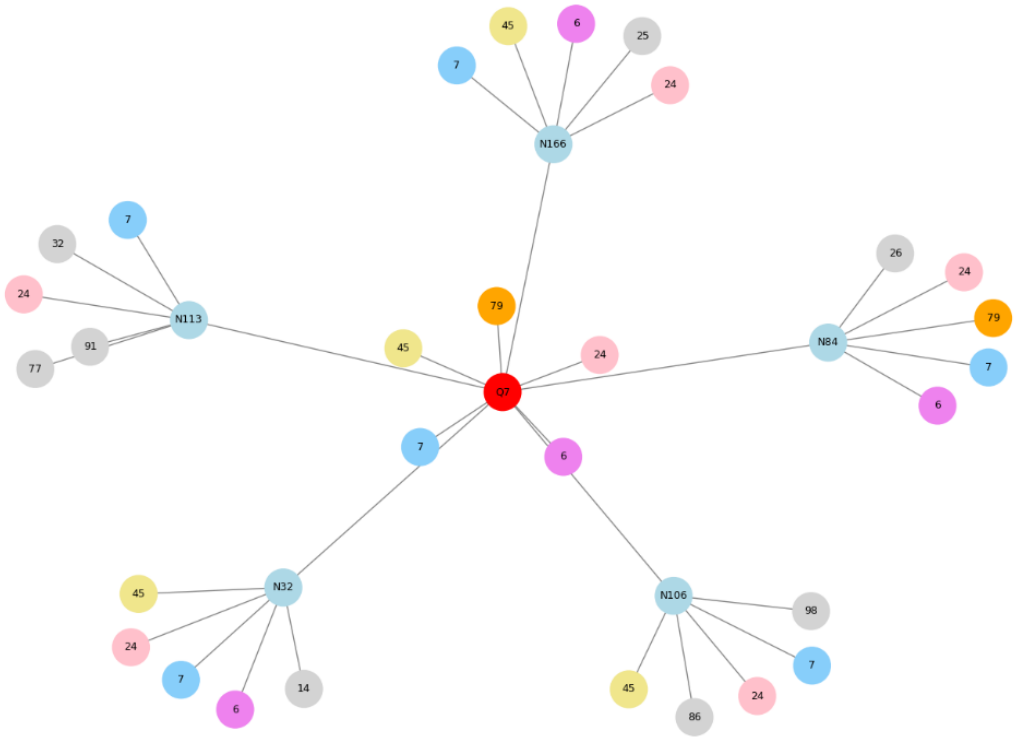

Figure 3.1: Example of selecting the class that appears most often among the neighbors and is simultaneously in the prediction of element 7. In this case, the class of element 7 will be 24 (pink).

Algorithm 1 summarizes the proposed solution.

Another solution was proposed based on the previous one. While in the last solution, the

Algorithm 1 Determining the k nearest neighbors using the cosine similarity between the features

Function `accuracy(`*cosine, output, target, topk = (1,)*`)`**:**

Input : cosine similarity matrix *cosine*, model logits *output*, ground truth labels *target*, tuple *topk*

Output: List of accuracy percentages for each k in *topk*

Step 1: Find nearest neighbors and initialize variables

knn indices $\leftarrow$ `Find Knn and Save(`*cosine, 5*`)` ; `// Get 5 nearest neighbors`

maxk $\leftarrow \max($*topk*$)$

batch size $\leftarrow$ size(*target*)

Step 2: Get model's top-k predictions

$,$ *pred* $\leftarrow$ topk(*output*, *maxk*, dim $= 1,$ sorted $=$ True)

Step 3: Collect neighbor predictions

neighbor preds $\leftarrow$ *pred*[*knn indices*]

Step 4: Majority voting for final prediction

final pred $\leftarrow [\,]$

for $i \leftarrow 0$ ***to*** *batch size* $- 1$ **do**

  all preds $\leftarrow$ *pred*$[i]$; `// Model's top-k for sample` i

  for $j \leftarrow 0$ ***to*** *neighbor preds.size*$(1) - 1$ **do**

    all preds.extend(*neighbor preds*$[i, j]$)

  end

  pred counts $\leftarrow \{\}$

  foreach $p \in$ *all preds* **do**

    pred counts$[p] \leftarrow$ *pred counts*.get$(p, 0) + 1$

  end

  final class $\leftarrow \arg\max($*pred counts*$)$

  final pred.append(*final class*)

end

Step 5: Combine predictions

combined pred $\leftarrow$ concat(*final pred*, *pred*.T$[: -1]$)

Step 6: Compute correctness

correct $\leftarrow$ (*combined pred* $==$ *target*)

Step 7: Calculate accuracy for each k

res $\leftarrow [\,]$

foreach $k \in$ *topk* **do**

  correct k $\leftarrow$ *correct*$[: k]$.sum$(0,$ keepdim $=$ True$)$

  res.append(*correct k* $\times\ 100.0 /$ *batch size*$)$

end

return *res*

algorithm of the k nearest neighbors was used in its original form, where all the neighbors have the same weight in the final classification vote, regardless of how similar or close the neighbors were to the *query* image, this solution changes that, making the neighbors most similar to the *query* have more influence, or more weight, in the final class decision.

Assigning weights proportional to the similarity between the *query* and its neighbors in the feature space is a natural and more informed extension of the classic K-NN algorithm. In this case, the closest neighbors in the latent space carry more relevant information about the class of the *query* than the most distant neighbors - even if they are all among the k closest.

Again taking the eighth element to exemplify this solution, its neighbors are the elements `[166, 32, 106, 113, 84]`, whose predictions are

```
[24, 7, 45, 79, 6, 24, 7, 6, 45, 25,
24, 14, 6, 7, 45, 24, 7, 45, 98, 86,
24, 32, 7, 77, 91, 79, 24, 6, 26, 7].
```

Everything up to this point is the same as the previous solution; these two solutions differ in the next step. In this case, in addition to considering the classes for the final prediction, a weight is also considered for each of these classes, depending on the number of times each class appears in the predictions of the neighbors of the element. For the eighth element, the table 3.1 shows the weights for each class.

Table 3.1: Weights of each class for the eight element for the second solution.

Classes	Weights
6	2.15168
7	3.54197
14	0.76414
24	3.54197
25	0.77860
26	0.60893
32	0.64378
45	2.28925
77	0.64379
79	0.60894
86	0.74651
91	0.64379
98	0.74651

In this particular case, there are two classes that have the same weight: class 7 and class 24. This was expected because there are five occurrences of these classes in the neighbors of the eighth element. The final prediction made by this model is class 7. Algorithm 2 summarizes the proposed solution, showing only the changes made in comparison with the Algorithm 1.

Algorithm 2 Weighted KNN Accuracy Calculation

Function `accuracy(`*cosine, output, target, topk = (1,)*`)`**:**

Step 1: Find nearest neighbors and initialize variables

knn indices, *knn sims* ← `Find Knn and Save(`*cosine, 5*`)` ; `// Addition: Store similarities as well`

Step 3: Collect neighbor predictions and similarities

neighbor sims ← *knn sims* ; `// Addition: Include neighbor similarities`

Step 4: Weighted voting (Modification)

weighted votes ← zeros(*batch size*, num classes) ; `// New weighted voting method`

for $i \leftarrow 0$ **to** *num neighbors* $- 1$ **do**

  current pred ← *neighbor preds*$[:, i]$

  current sim ← *neighbor sims*$[:, i]$

  weighted votes.add$(1,$ *current pred*, *current sim*$)$; `// Modification: Weighted addition`

end

Step 5: Final prediction using weighted voting

final pred ← argmax(*weighted votes*) ; `// Modification: Argmax from weighted voting`

Keeping in mind the idea of the k nearest neighbors algorithm, it was also thought of modifying this solution so that the number of neighbors to be considered in each *query* is determined dynamically. The idea behind this solution was to adapt the number of neighbors for each sample, based on their similarities, because certain samples may benefit from having a greater number of neighbors, while other examples may not.

The use of a dynamic number of neighbors in the K-NN algorithm is an evolution of the classic method, and its main objective is to make the inference process more sensitive to the context of the sample in the feature space. By dynamically adjusting the number of neighbors based on local similarities, the model can use fewer neighbors when the *query* is surrounded by highly similar examples, i.e. it is not necessary to *query* more points to make an accurate decision, or increase the number of neighbors when the closest examples show low similarity to the *query* image, where more context is needed for a more informed decision.

For example, for the eighth element, the neighbors considered in this case are as follows:

```
[7, 166, 32, 106, 113, 84, 146, 239, 261, 157]
```

The similarities of the neighbors are as follows:

```
[1, 0.78, 0.76, 0.75, 0.64, 0.61, 0.48, 0.47, 0.46, 0.42].
```

After dynamically determining the neighbors of each element, the final prediction is made as before. Figure 3.2 shows the final class decision for the eighth element.

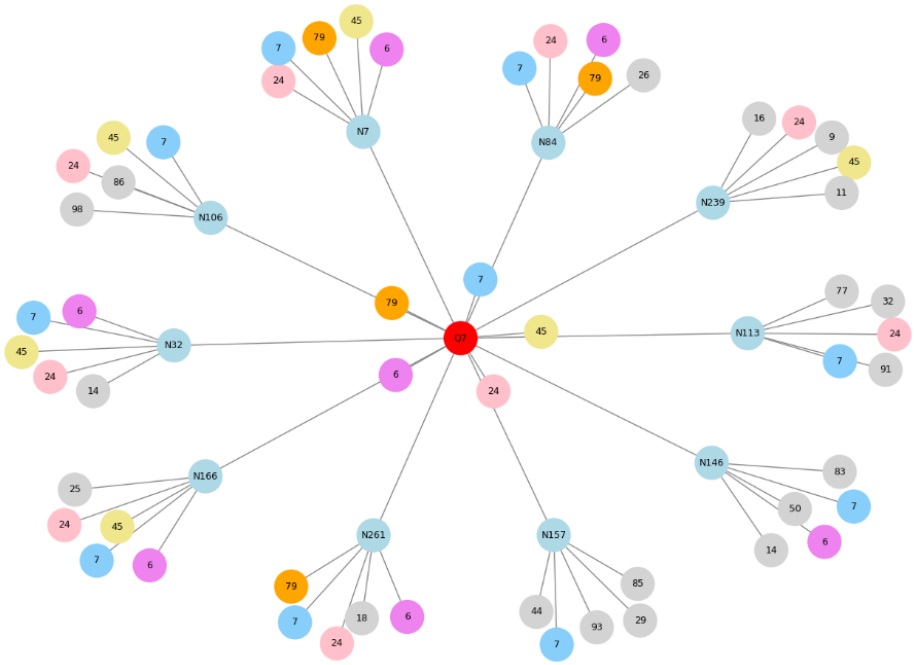

Figure 3.2: The class assigned to element 7 is 24 (pink), as it represents the most frequently occurring class among its neighbors and is also included in its predicted class set.

Algorithm 3 summarizes the proposed solution, showing only the changes made in comparison with the Algorithm 1.

Algorithm 3 Dynamic KNN Accuracy with Variable-Length Neighbors

Function `accuracy`(*cosine, output, target, topk = (1,)*)**:**

Step 1: Dynamic neighbor selection (Modification)

dynamic knn indices ← `find dynamic knn`(*cosine, max k=10, drop threshold=0.1*) ;

`// Modification: Dynamic neighbors instead of fixed k`

Step 4: Padding (New Step)

max neighbors ← $\max(\text{len}(n)\ \forall n \in \textit{neighbor preds})$

padded preds ← $\text{zeros}(\textit{batch size}, \textit{max neighbors}, \textit{maxk})$

Step 5: Final prediction refinement (Modification)

matches ← $(\textit{padded preds} == \textit{pred}).\text{sum}(\text{dim} = 2)$

final pred indices ← $\textit{matches}.\text{argmax}(\text{dim} = 1)$; `// Modification: Argmax based on dynamic voting`

final pred ← *pred*[*final pred indices*]

The last solution to the first gap was to change the k nearest neighbors algorithm so that the predictions of the neighbors are combined with their similarities. Although another solution to this problem already has this idea of giving different weights to neighbors with more or less influence, this new solution aims to use similarities as weights.

Combining the predictions of neighbors with their similarity levels represents a more informed and probabilistically based approach to improving the robustness and accuracy of

the K-NN algorithm. This new proposal differs from a previous solution by explicitly using similarity values as weights in the final decision, which guarantees a direct correspondence between confidence in the neighbor (based on proximity in the feature space) and its influence on the final classification.

As an example, as previously seen, the eighth element has the following elements as its neighbors `[166, 32, 106, 113, 84]`, whose similarities to the query element are as follows `[0.78, 0.764, 0.75, 0.64, 0.61]`. From the similarities it can be seen that the most similar neighbor of the eighth element is the neighbor with the index 166. Based on these neighbors and the weights of each neighbor, a weight is assigned to each class, and the class with the highest weight will be the final prediction for the element. In this specific case, and for the classes that are the predictions of the eighth element, their final weights can be seen in table 3.2. So, with this solution, the final class of this element is class 24.

Table 3.2: Weights of each class for the eight element for the fourth solution.

Classes	Weights
6	31.56564
7	66.52094
14	5.65591
24	131.98135
25	7.58910
26	9.28993
32	9.894956
45	33.40643
77	−7.40026
79	−7.32672
86	−5.06237
91	7.08871
98	20.16211

Algorithm 4 summarizes the proposed solution, showing only the changes made in comparison with the Algorithm 1.

Algorithm 4 Accuracy Computation with Reinforced KNN Voting

Function `accuracy`(*cosine, output, target, topk = (1,)*)**:**

Step 1: Reinforced KNN Voting (Modification)

knn pred, *weighted votes*, *neighbor indices*, *similarity scores* ← `reinforced knn voting`(*cosine, output*) ; `// Modification: Reinforced voting method`

Step 3: Fusion of predictions (Modification)

combined pred ← concat(*knn pred*, *model topk*.T[: −1]) ; `// Modification: Fusion between KNN and model predictions`

To mitigate the ambiguity that can occur between classes of the same superclass, due to their semantic similarities, this solution aims to fill the second gap by introducing a penalty in the loss function, with the aim of forcing the separation of classes within the same superclass.

Contrastive Learning for Visual Object Recognition

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i represents an input image and y_i its corresponding class label, the goal is to learn a representation function $f_\theta : \mathcal{X} \to \mathbb{R}^d$ parameterized by θ, such that:

- Representations of samples from the same class are close together in the latent space:

$$\forall i, j \quad y_i = y_j \Rightarrow \|f_\theta(x_i) - f_\theta(x_j)\| \text{ is minimized}$$

- Representations of samples from different classes are far apart:

$$\forall i, j \quad y_i \neq y_j \Rightarrow \|f_\theta(x_i) - f_\theta(x_j)\| \text{ is maximized}$$

This objective promotes intra-class compactness and inter-class separability in the learned embedding space.
The loss function with the introduction of the intra-class penalty can be seen in Equation 3.1.

$$\begin{aligned}\mathcal{L}_{\text{total}} = & -\frac{\tau}{\tau_{\text{base}}} \cdot \frac{1}{N} \sum_{i=1}^{N} \log \left(\frac{\sum\limits_{j \in P(i)} \exp\left(\frac{\text{sim}(z_i, z_j)}{\tau}\right)}{\sum\limits_{k \neq i} \exp\left(\frac{\text{sim}(z_i, z_k)}{\tau}\right)} \right) \\ & + \lambda_{\text{super}} \cdot \frac{\tau}{\tau_{\text{base}}} \cdot \frac{1}{N} \sum_{i=1}^{N} \log \left(\frac{\sum\limits_{j \in S(i)} \exp\left(\frac{\text{sim}(z_i, z_j)}{\tau}\right)}{\sum\limits_{k \neq i} \exp\left(\frac{\text{sim}(z_i, z_k)}{\tau}\right)} \right), \end{aligned} \quad (3.1)$$

where:

- τ is the temperature parameter,
- τ_{base} is the base (normalizing) temperature parameter,
- $P(i)$ is the set of samples from the **same class** as i,
- $S(i)$ is the set of samples from the **same superclass but different class** as i,
- λ_{super} controls the weight of the intra-superclass penalty,
- $\text{sim}(z_i, z_j)$ represents the similarity (scalar product) between embeddings.

Several plots were made to observe the distribution of the classes by their respective superclasses. Figure 3.3 shows this distribution, where it is possible to see some groups of data closer together, which supports this idea.
Algorithm 5 summarizes the proposed solution.

Algorithm 5 SuperclassContrastive Loss (ConLoss)

Class `ConLoss` *extends nn.Module*:

`// SelfContrastive Learning with Superclass Separation`

`__init__`(*temperature=0.07, contrastmode='all', basetemperature=0.07, lambdasuperclass=0.1, superclassmap=None*) `super().__init__()`

`forward`(*features, labels=None, mask=None, supcons=False, selfconsFG=False, selfconmFG=False*)

if $dim(features) < 3$ **then**
  `raise ValueError`
end
if $dim(features) > 3$ **then**
  $features \leftarrow features.\text{view}(features.\text{shape}[0], features.\text{shape}[1], -1)$
end
Batch size handling

$$batchsize \leftarrow \begin{cases} features.\text{shape}[0]/2 & \text{if } selfconmFG \\ features.\text{shape}[0] & \text{otherwise} \end{cases}$$

Mask preparation
if *both labels and mask provided* **then**
  `raise ValueError`
else if *neither provided* **then**
  $mask \leftarrow \text{eye}(batchsize)$
else if *labels provided* **then**
  $mask \leftarrow (labels == labels.\text{T}[:-1])$
else
  $mask \leftarrow mask$
end
$mask \leftarrow mask.\text{to}(features.\text{device})$
Superclass mask
if $self.superclassmap \neq None$ **then**
  $superlabels \leftarrow \text{map}(labels)$; `// Apply superclass mapping`
  $superclassmask \leftarrow (superlabels == superlabels.\text{T}[:-1]) - mask$
end
else
  $superclassmask \leftarrow \text{zeroslike}(mask)$
end
Feature preparation
$contrastfeature \leftarrow \text{concat}(\text{unbind}(features, \text{dim} = 1))$

$$anchorfeature \leftarrow \begin{cases} contrastfeature & \text{if } contrastmode = \text{'all'} \\ features[:, 0] & \text{otherwise} \end{cases}$$

Similarity calculation

$$logits \leftarrow \frac{anchorfeature @ contrastfeature.\text{T}[:-1]}{temperature}$$

$logits \leftarrow logits - \max(logits, \text{dim} = 1)$
Loss computation
$explogits \leftarrow \exp(logits) * logitsmask$
$logprob \leftarrow logits - \log(\text{sum}(explogits))$

$$loss \leftarrow -\frac{temperature}{basetemperature} \times \text{mean}(mask * logprob)$$

$$losssuper \leftarrow \frac{temperature}{basetemperature} \times \text{mean}(superclassmask * logprob)$$

$totalloss \leftarrow loss + lambdasuperclass \times losssuper$
return *totalloss*

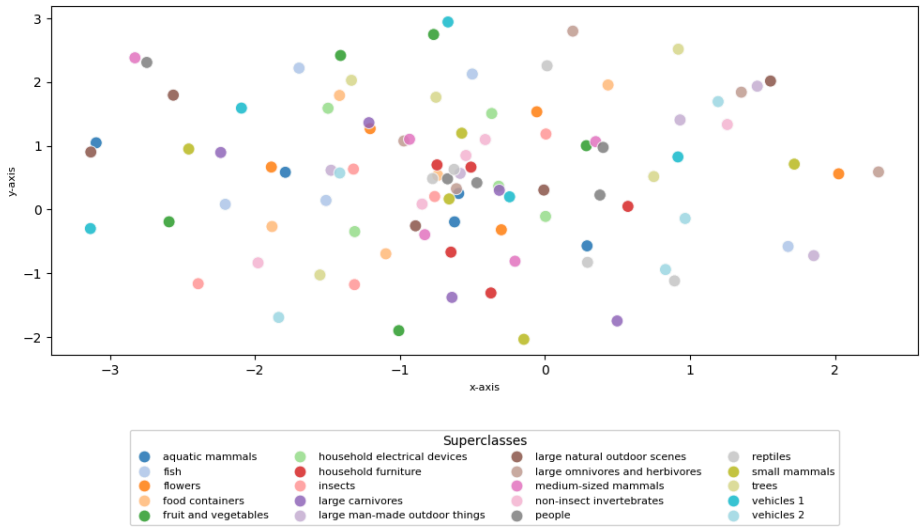

Figure 3.3: Distribution of the classes by their respective superclass.

3.4 Conclusion

This chapter explores the methodology that underpins the work carried out in this dissertation.

The problem and objective of this dissertation are explained in detail, followed by some limitations found in the method implemented. In addition, the various solutions implemented to fill these gaps are also explained. Each of these solutions has been explained briefly, followed by its algorithm and an example.

The next chapter provides more details on each of the solutions found, as well as explaining the set-up and implementation details.

Chapter 4

Implementation details

4.1 Introduction

This chapter presents the set-up and implementation details of the chosen method.
First, the datasets are presented, including the dataset used to train and test the method, the chosen method is presented, its parameters are explained, the original version of the method is explained and then the solutions found to combat the shortcomings explained in the previous chapter are explored in more detail.

4.2 Datasets

The different Contrastive Learning methodologies had something in common - the datasets used. Among the most common are the ImageNet and CIFAR-100 datasets.
ImageNet [39] is a dataset of images organized according to the WordNet hierarchy, where each node in the hierarchy is represented by thousands of images. This is one of the most important and fundamental datasets in the areas of computer vision and deep learning, and contains 14.197.122 images. The ImageNet dataset is highly regarded, but it is very large and training the different models with this dataset would be too time consuming. So it was decided to train the models with another dataset, also well known, the CIFAR-100. The CIFAR-100 [40] dataset consists of 100 classes with 600 images each. Of these 600 images, 500 images are used to train the model, and 100 images are used to test the model. This means that, in total, of the 60 k images, 50 k are used for training and 10 k are used for testing. This dataset does not have a validation set. The 100 classes in this dataset are grouped into 20 superclasses. Each image comes with a “fine” label, which corresponds to the class it belongs to, and a “coarse” label, which corresponds to the superclass it belongs to. The distribution of classes by their respective superclass can be seen in more detail in Table 4.1.

4.3 Method Implemented

Upon reviewing the methods detailed in Section 2.3, the most relevant techniques were selected for implementation. To do this, a research of the methods studied was carried out to

Table 4.1: Distribution of classes by their respective superclasses.

Superclass	Classes within the superclass
Aquatic Mammals	Beaver, Dolphin, Otter, Seal, Whale
Fish	Aquarium fish, Flatfish, Ray, Shark, Trout
Flowers	Orchids, Poppies, Roses, Sunflowers, Tulips
Food Containers	Bottles, Bowls, Cans, Cups, Plates
Fruit and Vegetables	Apples, Mushrooms, Oranges, Pears, Sweet peppers
Household Electrical Devices	Clock, Computer keyboard, Lamp, Telephone, Television
Household Furniture	Bed, Chair, Couch, Table, Wardrobe
Insects	Bee, Beetle, Butterfly, Caterpillar, Cockroach
Large Carnivores	Bear, Leopard, Lion, Tiger, Wolf
Large Man-Made Outdoor Things	Bridge, Castle, House, Road, Skyscraper
Large Natural Outdoor Scenes	Cloud, Forest, Mountain, Plain, Sea
Large Omnivores and Herbivores	Camel, Cattle, Chimpanzee, Elephant, Kangaroo
Medium-Sized Mammals	Fox, Porcupine, Raccoon, Skunk, Weasel
Non-Insect Invertebrates	Crab, Lobster, Snail, Spider, Worm
People	Baby, Boy, Girl, Man, Woman
Reptiles	Crocodile, Dinosaur, Lizard, Snake, Turtle
Small Mammals	Hamster, Mouse, Rabbit, Shrew, Squirrel
Trees	Maple, Oak, Palm, Pine, Willow
Vehicles 1	Bicycle, Bus, Motorcycle, Pickup Truck, Train
Vehicles 2	Lawn-mower, Rocket, Streetcar, Tank, Tractor

see which methods would give the best results. In addition, some of the methods studied were built on top of other methods, which means that they are expected to have better results. Table 4.2 shows the results obtained in **Top-1 Accuracy** by the main methods studied in this dissertation. All these results were taken from the respective articles, all these methods had the same network architecture (ResNet-50 [1]) and the results are shown for two important datasets, ImageNet and CIFAR-100 (although not all the methods had results in this dataset). The table also shows the results of the Cross-Entropy loss function for comparison, and these results are taken from the article [29].

Table 4.2: TOP-1 Accuracy results for the main CL methods

Method	Dataset	
	CIFAR-100	ImageNet
Cross Entropy	74.8%	76.5%
MoCo	–	60.6%
SimCLR	80.2%	69.3%
BYOL	78.4%	74.3%
SwAV	–	75.3%
Barlow Twins	–	75.3%
SupCon	76.5%	78.7%
SelfCon	78.5%	78.6%

[1] ResNet-50 [41] is a deep convolutional neural network architecture that was developed by Microsoft Research in 2015. This is a variant of the popular ResNet (or Residual Network) architecture, where the number 50 represents the number of layers in the network.

Implementing the studied methods was more challenging than expected, despite the availability of code on GitHub. Many recent works relied on outdated software versions incompatible with current hardware and systems. For instance, MoCo required multi-GPU training, making it impossible to run on a single GPU. Similarly, SwAV needed CUDA 10.1, which was unsupported on Windows 11. Two computers were used to test different methods, but conflicting dependencies meant that making one method work often broke another. After overcoming these obstacles, SelfCon was the only method that could be implemented with the available resources.

4.4 SelfCon Method

The SelfCon method was the method that was implemented, as previously mentioned. In addition to having this method implemented, the authors also implemented SupCon, which was very advantageous as there was no need to implement different one. Furthermore, the authors also implemented different versions of these methods - single-view and multi-view versions. A **multi-viewed framework** takes advantage of two factors, data augmentation and label information, while a **single-viewed framework** only uses label information. From now on, when a method has an *S* in front of it, then it is a single-viewed framework, and when it has an *M* in front of it, it is a multi-viewed framework.

4.4.1 Parameters

To implement the code provided by the authors, it was necessary to confirm that all the training parameters matched the parameters in the paper and on GitHub. Throughout the article in which the SelfCon method was developed, the various parameters that the authors used to train and test the model were presented. It is described that the **learning rate** used is 0.5. The following excerpts are taken from [29] :
"For small-scale benchmarks, we (...) set the batch size as 1024 for the pretraining and 512 for the linear evaluation. (...) We trained the encoder and the linear classifier for 1000 epochs and 100 epochs, respectively". "Every experiment used SGD with 0.9 momentum and weight decay of 1e-4 without Nesterov momentum. All contrastive loss functions used temperature τ of 0.1." Although these parameters are not directly defined, it is possible to find in the code the assignment of the arguments corresponding to the above.

```
parser.add_argument('--weight_decay', type=float, default=1e-4)
parser.add_argument('--momentum', type=float, default=0.9)
```

The temperature parameter is passed to the model when it is trained, so the value passed is 0.1, as indicated in the paper [29]. It is important to note that these parameters are specific to the CIFAR-100 dataset with a ResNet-18 network architecture. Throughout the paper, other parameters are described that are not taken into account, since other cases have not been implemented or tested. Table 4.3 briefly summarizes the parameters used for training and testing, as indicated in the paper and on GitHub.

Table 4.3: Parameters used to train and test the models.

Parameter	Value
Dataset	CIFAR-100
Network Architecture	ResNet-18
Learning Rate	0.5
Batch Size (Train)	1024
Batch Size (Test)	512
Epochs (Train)	1000
Epochs (Test)	100
SGD	0.9
Weight Decay	$1e-4$
Temperature	0.1

It is very important to confirm that these parameters are exactly the same as those used by the authors, so that the results are as reliable as possible. In addition to these parameters, there is a list of other arguments that are passed to the model that could take different values. Table 4.4 provides these arguments, an explanation of them, and the possible values they can take.

4.4.2 The Original Implementation of SelfCon

This subsection describes the original SelfCon framework as proposed by its authors.
The implemented method was carefully observed, specifically the part of the code where the accuracy of the model was determined, given that this is where the first observed gap is situated. The list below shows the step-by-step process carried out by the authors, explained in great detail:

1. The features are extracted using a Linear Classifier with a ResNet-18 architecture;
2. The `accuracy()` function computes accuracy over the top-k predictions, given:
    - The `output` tensor, `[batch size, number of classes]`, representing class probabilities;

[2]The Cosine Annealing Scheduler adjusts the learning rate following a cosine curve, so that it starts with a high learning rate that is decreased relatively quickly to a minimum value before being increased rapidly again.

Table 4.4: Arguments passed to the model, with their explanation and the values that each one can take.

Argument	Definition	Values that the argument can have
`Seed`	Ensures that results are reproducible	2022
`Method`	Indicates the training method	`Con`, `SupCon` and `SelfCon`
`Dataset`	Indicates the dataset that will be used (for training or testing)	`cifar10`, `cifar100`, `tinyimages`, `imagenet100`, `imagenet`
`Model`	Indicates the architecture of the model.	`ResNet, VGG, Wide ResNet, EfficientNet`
`Selfcon Pos`	Indicates the position where to attach the sub-network.	`Default: [False,True,False]`
`Selfcon Arch`	Indicates the architecture of the subnet	`resnet`, `vgg`, `efficientnet` and `wrn`
`SelfCon Size`	Indicates the number of blocks in the subnet.	`fc, same, small`
`Batch Size`	Number of training examples used in one iteration on training process.	$512, 1024$
`Learning Rate`	Determines the size of the steps the model takes when updating its weights during training.	0.5
`Epochs`	Number of complete passes through the training data.	$100, 1000$
`Cosine`	Indicates whether cosine annealing scheduling [2] should be used.	`True` or `False`
`Precision`	Indicates whether or not mixed precision should be used.	`True` or `False`
`Multiview`	Indicates whether to use multi-viewed batch.	`True` or `False`
`Label`	Indicates whether to use label information in the contrastive loss function.	`True` or `False`
`Temp`	Indicates the temperature used in the contrastive loss function.	0.07

- The labels, containing ground truth classes;
- A tuple `topk`, specifying the number of top predictions to consider;

3. The `pred` tensor stores the top-k predictions, in this particular case $k = 5$, (`[batch size, k]`), transposed to `[k, batch size]`;

4. A comparison tensor, `correct`, marks `True` for correct matches and `False` otherwise;

5. Accuracy values are stored in `res = []`, referring to the top-1 accuracy and the top-5 accuracy [3] (Acc@1 and Acc@5, respectively);

6. A loop iterates twice ($k = 1, 5$):

   6.1. Extracts the first k elements of `correct`;

[3] The top-1 accuracy is the conventional accuracy: the answer from the model (which has the highest probability of being correct), must be exactly the expected answer. The top-5 accuracy means that any of the five answers with the highest probability of being correct from the model are being considered.
For example, if we are dealing with an image whose class is a blueberry, and the predictions for a fruit are as follows: cherry - 0.35, raspberry - 0.25, blueberry - 0.2, strawberry - 0.1, apple - 0.06 and orange - 0.04, with top-1 accuracy, the prediction would be wrong, because the predicted class is not the actual class; with top-5 accuracy, the prediction would be correct, since the right answer is in the top-5 predictions.

   6.2. Reshapes and converts them to floating-point;

   6.3. Sums values to count correct predictions.

7. `correct_k` stores the correct predictions, converted to percentage, and subsequently added to `res`;

8. The function returns `res`, containing:

   - `res[0]`: Acc@1;
   - `res[1]`: Acc@5.

In addition, to combat the second shortcoming, it is necessary to understand how the loss is calculated, because this will be adapted to include the separation of classes. Within the `ConLoss` class, where the loss is calculated, the `__init__` function is initialized, which receives three parameters with different values:

- `temperature`: parameter that controls the smoothing of the `logits`, whose default value is 0.07;
- `contrast_mode`: defines whether all views (`all`) or just one (`one`) will be used as an anchor. The default is `all`;
- `base_temperature`: parameter used to normalize the loss to ensure numerical stability, whose default value is 0.07.

The `forward` function is then defined, where the loss is actually calculated. This function receives the as parameters the following:

- `features`: features tensor, where each sample has multiple augmented views (optional);
- `labels`: sample labels (optional);
- `mask`: matrix indicating which pairs are positive;
- `supcon_s`, `selfcon_s_FG`, `selfcon_m_FG`: flags for different variants of the contrast (the default for all these flags is `False`).

Within the `forward` function, the following is done:

1. The function starts by checking the size of the `feature` tensor and flattening extra dimensions if necessary. The batch size is set using its first position;

2. The `labels` and `masks` are processed:

- If both are provided, an error is raised;
- If neither is provided, an identity matrix of size `batch size` $\times$ `batch size` is created;
- If `labels` are given, a mask is generated, where `mask[i,j] = 1` if samples `i` and `j` have the same label, and 0 otherwise;

3. Anchors and contrasts are defined:
   - If `contrast_mode = one`: single anchor per sample;
   - If `contrast_mode = all`: every sample acts as an anchor, maximizing pairs;
4. `Logits`, representing similarity scores, are computed via the scalar product [4] and divided by `temperature`. The maximum value of each row is subtracted for numerical stability before softmax;
5. A `logits_mask` prevents self-contrast, ensuring positive pairs consist only of different views of the same class;
6. `exp_logits`, the normalized exponentiated logits, is derived:

   `exp_logits`$[i, j] = \exp(\text{similarity}(i, j))$ if $i \neq j$ or `exp_logits`$[i, j] = 0$ if $i = j$
7. Log-probabilities [5] are computed using:

$$\log p_{i,j} = \log \left(\frac{\exp\left(\frac{\text{sim}(i,j)}{T}\right)}{\sum_k \exp\left(\frac{\text{sim}(i,k)}{T}\right)} \right) \quad (4.1)$$

   where $\text{sim}(i, j)$ is the similarity score between the anchor i and the contrast j and T represents the `temperature`. So, for each anchor i, `log_prob`$[i, j]$ is the log-probability that contrast j is the correct positive given anchor i;
8. `mean_log_prob_pos` captures only the positive contrasts, ensuring the focus remains on relevant samples:

[4] The scalar product, also known as dot product, is a way to measure how similar two vectors are. It takes two vectors and gives back a single number (a scalar). It is calculated like this: $\text{dot}(a, b) = a_1b_1 + a_2b_2 + \cdots + a_nb_n$, where a_n is the n-th element of the vector a.

[5] The log-probabilities, as the name indicates, are the logarithm of a probability. Instead of working directly with the probabilities, their log is often used because it is more stable, especially when multiplying many together, and it is easier to work with mathematically.

$$\text{mean\_log\_prob\_pos}_i = \frac{1}{|P(i)|} \sum_{j \in P(i)} \log p_{i,j} \quad (4.2)$$

$P(i)$ represents the set of positive samples for anchor i, which means that $|P(i)|$ is the number of positive samples; $p_{i,j}$ is the probability of the contrast j being a correct match for the anchor i;

9. The loss is calculated by multiplying the `mean_log_prob_pos` for a scaling factor (based on the `temperature` and `base_temperature`), and then negating it (this is a negative log-likelihood, so lower is better). The loss is then reshaped and it takes the mean over all anchors.

4.5 Implemented Solutions to Mitigate the Identified Gaps

4.5.1 Determining the Five Nearest Neighbors based on the Cosine Similarity of the Features

The first solution to be implemented was to determine the five nearest neighbors based on the cosine similarity [6] of the features. The idea behind this change is to improve Acc@1. By finding the five closest neighbors of an element and basing these neighbors on the cosine similarity of the features, it is possible to use the predictions of the classes of the neighbors to make a better decision regarding the class of a given image. By having features similar to those in the query image and their predictions, the percentage of choosing the right class should, in theory, be higher.
To implement this solution, two new functions have been added:

- `def cosine_similarity(features)`

This function receives only one parameter, in this case, the tensor of the features. As the name of the function indicates, the cosine similarity of the features is calculated here. A matrix is created, of dimension `[len(features), len(features)]`, where, at position `similarity_matrix[i,j]`, its value is the cosine similarity between `feature[i]` and `feature[j]`. If $i = j$, its value will be 1. Below is an example of the similarity matrix.

[6] Cosine similarity quantifies the similarity between two non-zero vectors within an inner product space. This measure produces a value ranging between 0 and 1. A value near 0 indicates that the vectors are orthogonal or perpendicular, indicating minimal similarity. In contrast, a value approaching 1 implies a smaller angle between the vectors, indicating greater similarity.

$$\begin{bmatrix} 1.0000 & 0.2240 & \cdots & 0.2100 & 0.1906 \\ 0.2240 & 1.0000 & \cdots & 0.1780 & 0.2679 \\ 0.2745 & 0.2363 & \cdots & 0.2267 & 0.1629 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0.2549 & 0.2372 & \cdots & 0.2834 & 0.2855 \\ 0.2100 & 0.1780 & \cdots & 1.0000 & 0.2323 \\ 0.1906 & 0.2679 & \cdots & 0.2323 & 1.0000 \end{bmatrix}$$

The other function that was implemented was the following:

- `def find_knn_and_save(similarity_matrix, k, filename="knn_results.csv")`

This function receives three parameters, the `similarity_matrix`, a tensor representing the cosine similarity between features, k, the number of nearest neighbors to consider and `filename`, the name of the output CSV file, where some information about the model is stored, used for example purposes.
The objective of this function is to determine which are the closest neighbors to each of the features, so k represents the number of neighbors we want to determine. The calculation of the k nearest neighbors of each feature is based on the similarity matrix, calculated with the help of the previously explained function, `cosine_similarity`. The k highest similarity values for each feature are its k nearest neighbors. Note that since the similarity of a feature to itself is 1, this value is not taken into account, as it makes no sense to consider the value as its own neighbor.
This function determines the index of the nearest neighbors of each feature.
These are the two functions that have been introduced. The list below shows the step-by-step of the `accuracy` function, now with the new functions and the changes made:

1. The features are computed using a Linear Classifier, yielding the `output` tensor;
2. The `output` tensor is passed to the `accuracy` function along with labels and $topk = (1, 5)$;
3. The modified `accuracy` function now also receives the similarity matrix, `similarity_matrix`, obtained through `cosine_similarity()`.
4. The function `find_knn_and_save()` determines the 5 nearest neighbors, and stores their indices in `knn_indices`;

5. The batch size and max `topk` value are extracted, then top-k predictions are computed and stored in `pred`;
6. The neighbor predictions are stored in `neighbor_preds`;
7. `final_pred` is initialized to store the most frequent class among neighbors;
8. For each sample `i`:
   8.1. All the predictions of the i element and its neighbors are collected;
   8.2. All the predictions are combined into a list, `all_preds`.
   8.3. The frequency of each class in `all_preds` is counted;
   8.4. The most frequent class is selected as the final prediction, breaking ties by first occurrence;
   8.5. The final predictions are stored in `final_pred`;
9. `final_pred` is converted to a tensor.
10. The new predictions replace the last model prediction in `pred`;
11. As before, Acc@1 and Acc@5 are calculated to evaluate performance.

4.5.2 Weighted k-Nearest Neighbors

The last solution, presented in subsection 4.5.1, implemented the typical neighbor determination scheme, i.e. all the neighbors have the same weight in the final prediction. However, this may not be the most suitable solution, since there are some neighbors that will be more similar to the *query* than others, which means that the final prediction may not be the most accurate. The idea behind the second solution was then to weight each vote of the neighbors based on its similarity to the *query* in question. Neighbors that are more similar to the current sample contribute more (have more influence) in the final prediction, while less similar neighbors have a lower weight in the prediction. The list below shows the step-by-step process behind implementing this solution:

1. The `accuracy` function takes the similarity matrix, `output` tensor (logits), `target` tensor (true labels), and `top-k` tuple for the accuracy calculation;
2. The five closest neighbors and their similarity scores are obtained through the `find_and_save()` function;

3. The `pred` tensor stores the top-5 predictions;
4. The neighbor predictions are collected in `neighbor_pred`;
5. The `weighted_votes` tensor is initialized to store similarity-weighted votes;
6. Each sample iterates over its five nearest neighbors:
   6.1. The predictions of the neighbors are collected;
   6.2. The similarity scores are also collected;
   6.3. The votes are assigned based on similarity weighting;
7. The most frequent similarity-weighted class is selected as the final prediction, resolving ties by first occurrence;
8. `combined_pred` combines weighted votes with model predictions;
9. The `correct` tensor verifies prediction accuracy;
10. Acc@1 and Acc@5 are computed as in previous methods.

4.5.3 Dynamic k-Nearest Neighbors Prediction

The next solution was also made on the basis of nearest neighbors according to cosine similarity. However, instead of always selecting the same number of neighbors (as done previously, where five neighbors were selected), the number of neighbors is selected dynamically. This selection of neighbors is based on the similarity of the neighbors and a threshold previously established as the difference. The idea behind this solution was to try to improve the result of some elements whose prediction could be improved with the help of their neighbors, while elements that were not so similar to their neighbors were not harmed by this. This idea of determining the neighbors dynamically took several iterations, i.e. small adjustments were made to the initial idea to see how the model would behave. The different iterations are described below. The first iteration is described in more detail, and the following iterations only describe the adjustments that were made.

First Iteration

The first iteration was the basis for the following ones. For all iterations, it was necessary to calculate the neighbors dynamically, and for this a new function was created:

- `find_dynamic_knn(cosine_similarities, max_k=10, drop_threshold=0.1, filename="neighbors_info.csv"):`

This function receives four parameters: the `cosine_similarities`, a tensor that represents the cosine similarity matrix, `max_k`, the maximum number of neighbors to consider (`default:`10), `drop_threshold`, the threshold to detect a drop/difference in the similarity of the examples (`default:`0.3)[7] and `filename`, the name of the CSV that will store information about the neighbors (just to visualize how the method is working and to show examples).
In order to calculate the neighbors dynamically, the first step is to order the indices of the neighbors in terms of similarity for each sample. The `sorted_indices` tensor contains this information, where the neighbors of each element are ordered from most to least similar.
For each element, the following is done:

1. The `sims` tensor stores similarity values, sorted from highest to lowest;
2. The differences between consecutive similarities are computed;
3. Values exceeding the threshold are removed, adjusting the value from 0.1 to 0.3 to include more neighbors;
4. The number of neighbors is set as the minimum of `max_k` (10) or the available neighbors, with a default if none are found;
5. Final neighbor indices and similarities are collected.

The function returns the indexes of the dynamically determined neighbors. This function now needs to be integrated with the `accuracy()` function. In a very brief way, since many of the steps are similar to the ones done in other solutions, the following is done:

1. The neighbors for each sample are determined, with the help of the `find_dynamic_knn()` function described above;
2. As is always done, the top-k (top-5) predictions of the model are determined;
3. The predictions of each neighbor are collected;
4. The final prediction is made as was done in alteration 1, with a count of the classes predicted by the neighbors and the class with the most votes being chosen;
5. This prediction is combined with the predictions of the model;
6. Acc@1 and Acc@5 are calculated, and these results are returned.

[7] If the value is lower, in theory, fewer neighbors will be considered, since the similarity has to be high for them to be considered important

Second Iteration

In the previous iteration, the number of neighbors was determined as the minimum between $\texttt{max\_k} = 10$ and the number of neighbors found. If no similar neighbors were identified based on the threshold, the default value of 10 was used. However, this compromised the core idea of ensuring the only the most similar neighbors influenced the final prediction. The second iteration refines this by ensuring that if an element has no neighbors, the default value is not used - instead, the element is treated as its own neighbor. This adjustment is not necessarily better or worse than the previous method; rather, it aims to test whether excluding the default setting leads to any improvement.

Third Iteration

In the third iteration, the approach to determining the final class prediction was modified. Previously, the prediction was based on the neighbor with the most matches to the sample. Now, instead of relying on individual neighbors, the method counts the total occurrences of each predicted class among all neighbors. The class with the highest count is chosen as the final prediction. In case of a tie between multiple classes, priority is given to the first prediction in the top-k predictions of the sample.

4.5.4 Reinforced k-Nearest Neighbors

This solution uses cosine similarities as weights when combining with the predictions of the k nearest neighbors. The main aim of this solution is to improve the quality of the final predictions by giving more weight to neighbors that are more similar to the current sample. In order to implement this new solution, a new function was created:

- `reinforced_knn_voting(cosine_similarities, output, k=5):`

This function receives three parameters, the `cosine_similarities`, a tensor representing the cosine similarity matrix, the `output`: the output of the model for each sample and $\texttt{k} = 5$, the number of neighbors to be considered.
This function does the following:

1. The k nearest neighbors and their similarities are obtained using the cosine similarity matrix;

2. The model outputs for the five nearest neighbors are extracted using `output`;

3. Each output of the neighbors is weighted by its similarity, with votes aggregated to compute a final weighted score per class;
4. The class with the highest cumulative weight is selected as the final prediction;
5. The function returns final predictions, weighted scores per class, neighbor indices, and similarities.

This new function had to be integrated into the `accuracy()` function. To summarize, the following is done:

1. The similarity-weighted neighbor predictions are determined to obtain the final k-NN prediction for each sample, using the `reinforced_knn_voting` function;
2. As in the previous cases, the top-k predictions are determined from the output of the model;
3. The weighted prediction is combined with the top-k predictions of the model;
4. The predictions made are compared with the ground truth labels to check that each prediction is correct;
5. Acc@1 and Acc@5 are calculated, and these results are returned.

4.5.5 Forcing the Separation of Classes within a Superclass

Motivated by the dataset used, and the way the classes and superclasses were divided, the next solution aims to address the second identified gap. As seen earlier, the CIFAR-100 dataset is divided into 100 classes, which can be combined into 20 superclasses. For more details on this division, please refer to section 4.2. This division of the classes into their respective superclasses raised a question about the classification being done by the model - were the classes being confused within the same superclass? As there are different superclasses, which are sort of broader groups that contain different classes, which may have similarities between them, then something the model could be doing is confusing these classes within the superclass. For example, the classes `Poppies` and `Tulips` belong to the superclass, `Flowers`, and, as they are both flowers, of course they have similarities and could therefore be confused with each other. The idea is then to check how the classes were being classified in relation to their superclass and to see if there could be any swapping of classes within the same superclass. To help visualize this classification, different plots were made showing the distribution of the 100 classes and their division into the 20 superclasses. If the dots of the

same color on the plot (which indicate classes that belong to the same superclass) were very close together, this would indicate that the classes within the same superclass might be getting confused. It would then be interesting to force a separation in the classes that belonged to the same superclass, and to see if, by forcing this separation, the results and classifications improved. This separation would be forced in the loss function and, to do this, it was necessary to change the initial loss code. Please refer to subsection 4.4.2 to review the initial implementation of the loss function. Many steps of this initial implementation are preserved when a separation of classes within a superclass is introduced, but there are some differences between the strategy of the authors and the new strategy, which will be explored bellow. The main difference between the two versions is the introduction of the class separation within a superclass. This modification penalizes samples that belong to the same superclass and are from different classes within that same superclass. The parameter `lambda_superclass` is introduced, which adjusts the weight of the penalty within each superclass. A dictionary is also initialized, the `superclass_map` dictionary, that maps each class to its respective superclass. After the definition of the standard mask for the classes (`mask`), discussed above in Step 2, this new version of the loss function also generates a mask for the superclasses.
The mask `superclass_mask` is defined as:

- 1, for elements that belong to the same superclass but have different classes;
- 0, otherwise.

This is one of the factors that helps to better separate samples from the same superclass. In addition to the original contrastive loss, the new version of the loss function also calculates:

- The average of the log-probabilities for pairs from the same superclass, but from different classes;
- A penalty for these samples, which encourages separation within the same superclass.

The original final loss is adjusted to include this intra superclass penalty.

4.6 Conclusion

The main methods of Contrastive Learning have been reviewed with the aim of trying to implement a novel method. The implemented method, SelfCon, was analyzed in detail, as well as the parameters associated with the method, with a brief explanation of each.
The intermediate subsections focus on the solutions made to address the problems found, starting with an explanation of the method proposed by the authors, followed by various

solutions that were developed with the intention of improving the performance of the model and overcoming the shortcomings.

The next chapter presents the results obtained, starting with the base results of the model, and also presenting the results of the solutions.

Chapter 5

Results

5.1 Introduction

This chapter initially addresses the preliminary results obtained by the original method, and also discusses some changes made to the parameters to improve the test time of the model. The results obtained by the different solutions made to the model will then be analyzed in detail, comparing the initial results with the new ones and drawing some general conclusions about the results.

5.2 Preliminary results

5.2.1 Training and Comparison of Results

As mentioned above, the method implemented was SelfCon. Once all the requirements for the code to work had been met and all the parameters had been checked, it was necessary to test the code to check that the results obtained were the same as those presented in the article. Table 5.1 contrasts the original results with the results of the replicated experiments. To ensure efficiency in the training process, the model was trained with the CIFAR-100 dataset instead of the ImageNet dataset. The results presented are based on the ResNet-18 model.

Table 5.1: Comparison between the Top-1 Accuracy results of our training and the results presented in the article.

Method	Results	
	Ours	Authors
SelfCon-S	74.9%	75.4%
SelfCon-M	74.38%	74.9%
SupCon-S	74.41%	73.9%
SupCon-M	72.68%	73%

It should be noted that the results presented here were obtained by following the steps and parameters provided by the authors of the method on their GitHub. Four different methods were trained: SelfCon-S, SelfCon-M, SupCon-S and SupCon-M. In general, the results obtained when testing these models were slightly worse than those reported by the authors, with the exception of the SupCon-S method. However, when a change was initially proposed

and this first implementation was not implemented efficiently, testing the changes was taking a long time. To reduce this time, the model was tested with two changes to the parameters:

- Decreasing the batch size: as the initial implementation of the cosine similarity matrix was not efficient, calculating the matrix took a long time, and the batch size also affected this calculation (initially a 512×512 matrix was calculated). So, to improve this time, the batch size was reduced to 128;
- Removal of the sub-network: the model was tested on the main network and also on a sub-network, which doubled the training time, and so it was removed.

To compare the effects of these removals with the results obtained when training the model with the initial parameters, the four methods were tested again, and the results of these tests can be seen in table 5.2.

Table 5.2: Comparison of the results obtained by testing the model with the initial parameters and changing the parameters.

Methods	Original Results	Simplified Version Results
SelfCon-S	74.90%	74.39%
SelfCon-M	74.38%	74.36%
SupCon-S	74.41%	72.73%
SupCon-M	72.68%	72.51%

Figure 5.1 also helps to see the change in Acc@1 over the course of testing the model with the initial parameters and with changes to them.

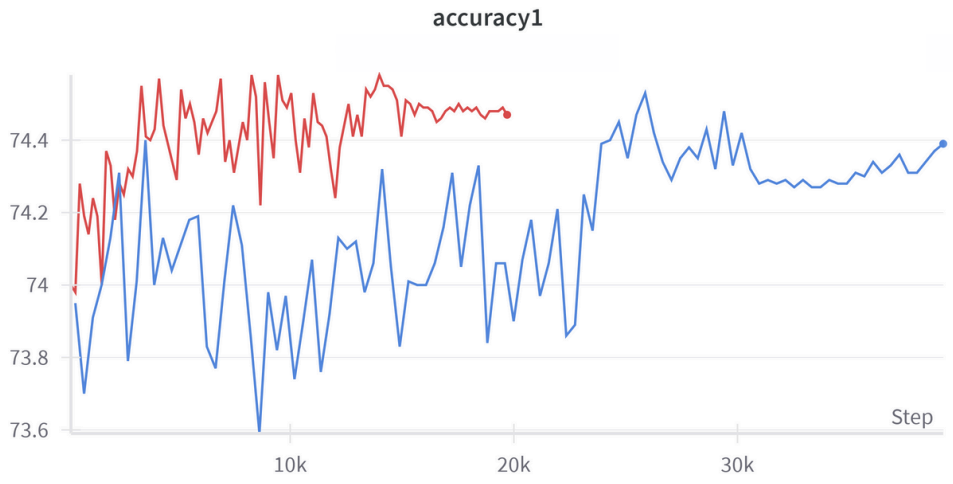

Figure 5.1: Comparison of Acc@1 between the model trained with the initial parameters (red) (batch size 512 and with subnet) and the model trained with batch size 128 and without subnet (blue) (Method: SelfCon-S).

When the batch size is 512, the final Acc@1 is 74.47; when the batch size is 128, the final Acc@1 is 74.39, which means that the difference between them is only 0.08. In addition, the

test time also decreased with the change made to the parameters. Note that when the batch size is smaller (the blue line in the graph), more iterations are made over the data, so the line is longer (more steps are taken). When the batch size is larger (the red line in the graph), fewer steps are taken. Since the difference between the results is minimal and the time the model takes to test is shorter, all the tests from now on will be done with these changes to the parameters (with the batch size at 128 and without sub-net).

5.3 Results

5.3.1 Determining the Five Nearest Neighbors based on the Cosine Similarity of the Features

This solution consisted of determining the five closest neighbors of a *query* based on the cosine similarity of the features, and creating a new final prediction based on these neighbors, as described in detail in Subsection 4.5.1. The results of this solution are summarized in Table 5.3.

Table 5.3: Acc@1 results after the implementation of this solution.

Methods	Base Results	Results
SelfCon-S	**74.39%**	71.65%
SelfCon-M	74.36%	70.67%
SupCon-S	72.73%	68.23%
SupCon-M	72.51%	65.10%

The implementation of this solution had a negative impact on Acc@1. This may suggest that this selection of neighbors using cosine similarity may not provide important information for the prediction of the model compared to the approach taken by the authors. One possible explanation is that the feature space learned by the model does not organize instances in a way that cosine similarity reflects semantic similarity between samples. Within these results, the method that had the most impact was the SupCon method, regardless of whether it was in single-view or multi-view settings, which may indicate that perhaps, in this particular method, the use of neighbors and cosine similarity is not the most suitable choice, as the predictions worsened. It is also important to note that the methods with the multi-viewed framework show a more significant reduction. In addition to the observations, it is also important to note that the SelfCon method, in both settings, had the best results (although they dropped with Alteration 1), which is in line with the initial results. Another aspect that can also be seen in the results, and which is also consistent with the initial results, is that the single-viewed framework has the best results in both methods. The difference between the

single-viewed framework and the multi-viewed framework is also greater after the change has been made, which may suggest that the latter type of setting does not adapt as well to the introduced change.

5.3.2 Weighted k-Nearest Neighbors

This solution consisted of weighting each vote of the neighbors based on their similarity to the query in question. Neighbors that are more similar to the current sample contribute more (have more influence) in the final prediction, while less similar neighbors have a lower weight in the prediction, as described in detail in Subsection 4.5.2. The results of this solution can be found in Table 5.4.

Table 5.4: Acc@1 results after implementing this solution.

Methods	Base Results	Results
SelfCon-S	**74.39**%	51.52%
SelfCon-M	74.36%	53.11%
SupCon-S	72.73%	53.20%
SupCon-M	72.51%	51.20%

These were the weakest results across all the applied solutions. All the results of the methods decreased by around 20%, a much more significant difference compared to the previous solution, where the difference was at most around 7%. By assigning greater influence to neighbors considered more similar based on cosine similarity, the method assumes that these similarity scores reliably indicate true relevance to the class of the query. However, as previously discussed, cosine similarity may not accurately capture semantic closeness in the learned feature space. The results obtained here also seem to contradict the baseline results. Methods with a single-viewed framework initially show better results than the methods using a multi-viewed framework, but this is not maintained after the alteration. In this case, in the SelfCon method, the multi-view setting has better results that the single-view setting, and in the SupCon method, the opposite is true. Furthermore, the differences between the methods with a single-viewed framework and with a multi-viewed framework are around two percentage points, unlike the initial results, in which the difference between the frameworks was only tenths.

5.3.3 Dynamic k-Nearest Neighbors Prediction

This solution was the determination of the number of neighbors which is selected dynamically, instead of having a fixed number of neighbors defined. This selection of neighbors is based on the similarity of the neighbors and a threshold previously established as the dif-

ference. As it was seen in Subsection 4.5.3, three iterations were made within this solution. The first iteration was the main one, on which the other two iterations were built. All the iterations were tested separately and their results were observed, and something quite interesting happened - the second and third iterations, despite changing the initial iteration, have no weight in the decision of the model. The truth is that, after testing the model with these changes, the results proved that these iterations do not matter at all. The outcomes were nearly identical to those of the first iteration (differing only by thousandths), making further analysis unnecessary. So, since this happens, the results to be considered will only be those from the first iteration. The results can be seen in Table 5.5.

Table 5.5: Top-1 accuracy results after applying this solution.

Methods	Base Results	Results
SelfCon-S	**74.39%**	74.37%
SelfCon-M	74.36%	74.30%
SupCon-S	72.73%	72.70%
SupCon-M	72.51%	72.50%

This solution had results that were very similar to the initial results. This suggests that, perhaps with more significant changes to the method introduced, this solution could improve the base results. The results obtained here are exactly in agreement with the initial results, which makes sense as the results are very similar. The single-viewed framework have the best results in both the SelfCon and SupCon methods, and, as was initially the case, the SelfCon method has the best results in both frameworks.

5.3.4 Reinforced k-Nearest Neighbors

This solution explores the reinforced k-nearest neighbor approach, using cosine similarities as weights when combining the predictions of the neighbors, as described in detail in Subsection 4.5.4. The results of solution can be seen in Table 5.6.

Table 5.6: Acc@1 scores following the implementation of this solution.

Methods	Base Results	Results
SelfCon-S	**74.39%**	72.37%
SelfCon-M	74.36%	71.55%
SupCon-S	72.73%	69.62%
SupCon-M	72.51%	66.73%

Looking at the obtained results, several conclusions can be drawn. First of all, the methods that used a single-viewed framework had the best results compared to the multi-viewed framework. In addition, the SelfCon method obtained the best results overall. These two points are consistent with the base results. Comparing these results with the first solution,

seen in Table 5.3, these results are better. This is already an improved version of that solution, whose results are already better, and, perhaps with other alterations, the results could improve even more, possibly surpassing the base results.

5.3.5 Forcing the Separation of Classes within a Superclass

This solution forced classes belonging to the same superclass to move apart, so that classes within each superclass would not be confused, as described in detail in Subsection 4.5.5. The results can be found in Table 5.7.

Table 5.7: The obtained Acc@1 metrics after applying this solution.

Methods	Base Results	Results
SelfCon-S	74.39%	**74.91%**
SelfCon-M	74.36%	74.21%
SupCon-S	72.73%	73.69%
SupCon-M	72.51%	72.18%

Among all tested solutions, this was the sole solution that demonstrated a measurable improvement over the baseline results. In particular, the improved results were made in single-viewed frameworks. Both methods in this type of framework had their results improved, which shows that this is a good strategy for improving the Acc@1 of the methods. The SupCon-S method had the biggest increase, with around one percentage point compared to the base result.

The SelfCon-S method also exhibited an increase, albeit a smaller one.

The results of these two methods are in line with the base results: the methods in single-view settings have better results than the methods in a multi-viewed framework. In addition, the multi-viewed framework methods also obtained results that were very similar to the base results, which suggests that better results can also be achieved in this setting, with some changes that may be negatively affecting these specific methods. To better visualize the results obtained here, plots were made comparing the distribution classes by superclass before and after applying this method, to see how this distribution changed.

For all the figures below, the following is applicable:

- There are one hundred points in the figure, divided between the twenty existing superclasses. Each superclass, represented by a different color, has five point associated with it, which represent the different classes belonging to the same superclass;
- The figures below have different axes, as initially the points were much more grouped together, and after the fifth modification, they were much further apart - the difference in scale shows this.

Figures 5.2 and 5.3 show the difference in the SelfCon-S model before and after this solution.

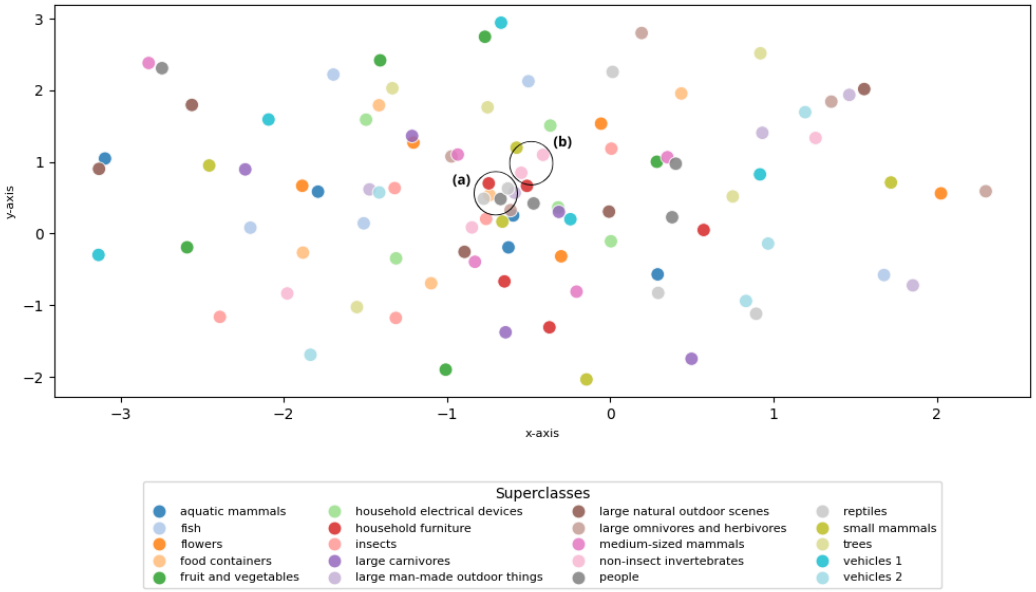

Figure 5.2: SelfCon-S method before implementing this solution.

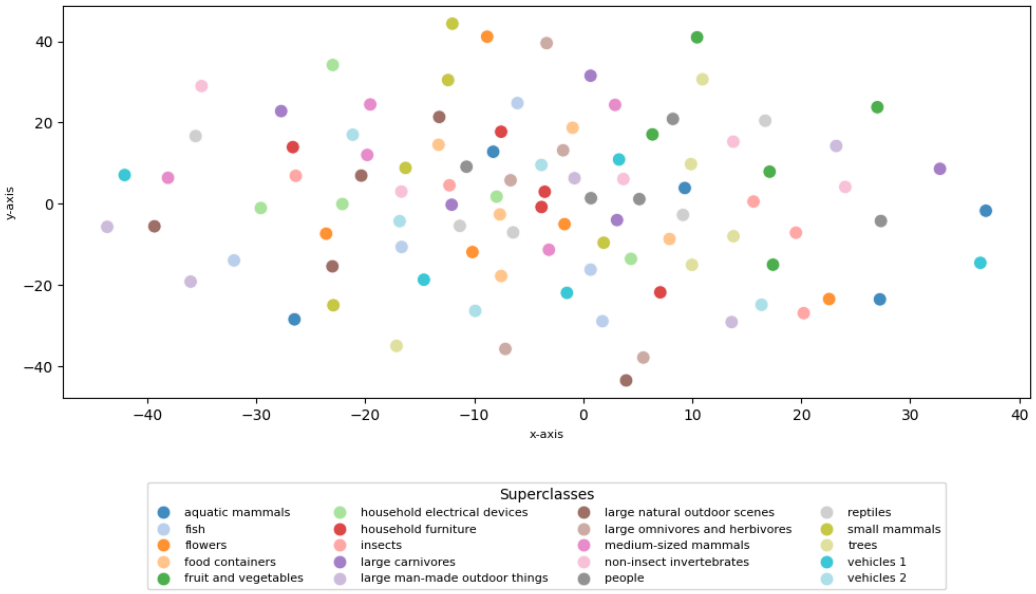

Figure 5.3: SelfCon-S method after implementing this solution.

Figure 5.2 shows the SelfCon-S method before the application of this solution. It can be seen that the points in this figure are very clustered. Also in this figure, the following can be highlighted:

(a) In the circle labeled as `(a)`, it can be seen that two of the classes belonging to the `reptiles` superclass (identified in light gray in the figure), are quite close;

(b) In the circle identified as `(b)`, it can be seen that there are two points quite close together, which belong to the superclass `non-insect invertebrates` (in pastel pink in the figure).

Despite the fact that the points are quite closely grouped, the initial method already had good results, since there is not much overlap between the superclasses. In figure 5.3, the points representing the superclasses are much further apart. It can be seen that all the points have moved away from each other, which was to be expected. However, compared to the initial case, there are two red dots belonging to the `household furniture` class that seem to have moved closer together. Moreover, it is clear from viewing the plot that the points representing the different classes within the respective superclass are much further apart. Figures 5.4 and 5.5 show the plots for the SelfCon-M method before and after the implementation of this solution.

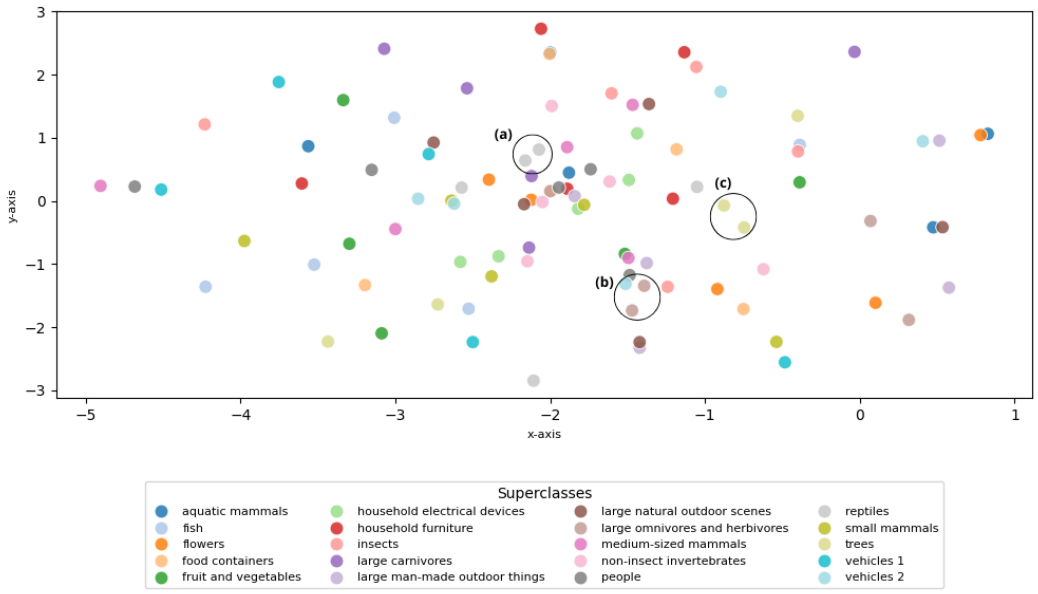

Figure 5.4: SelfCon-M method before implementing this solution.

In the figure 5.4 it can again be seen that the points are very closely grouped, and the following can be highlighted:

(a) In the circle identified as `(a)`, the `reptiles` superclass has two points very close to each other in the center of the plot, with a third point considerably close to these two on the left;

(b) In the circle identified as `(b)`, the `trees` superclass (represented in pastel green in the figure) also has two points very close together, which may suggest that these two distinct classes are being confused within the same superclass;

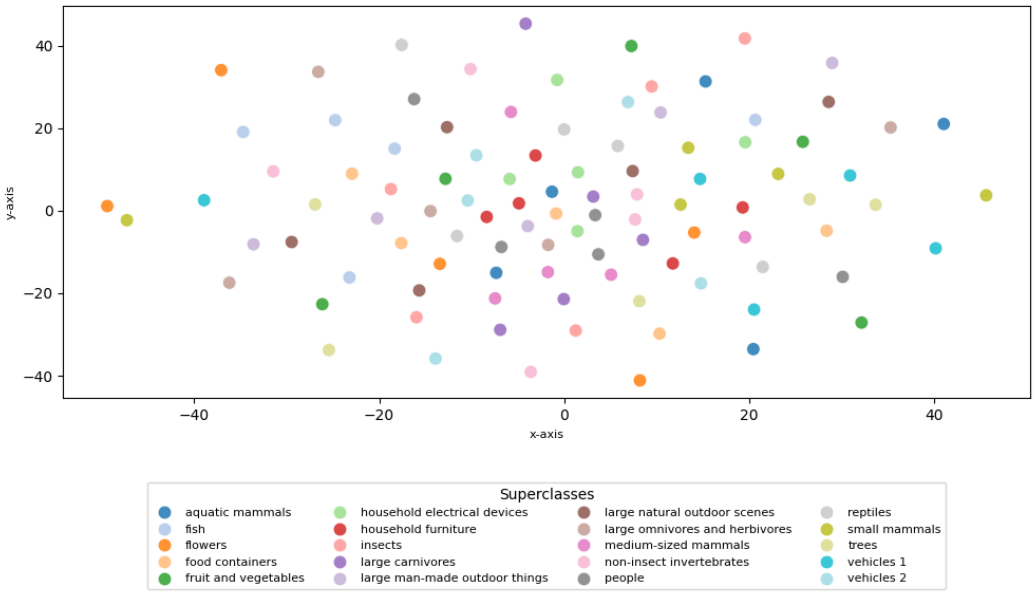

Figure 5.5: SelfCon-M method after implementing this solution.

(c) In the circle identified as `(c)`, the `large omnivores and herbivores` (identified in light brown in the figure) also has two points close together.

These are just specific examples, where classes (points in the figure) of the same superclass are closer together, but in the figure there are other points that are also relatively close together. In the figure 5.5, it can be seen a noticeable distance between the points belonging to the same class. Although, in this particular case, the Acc@1 results have not improved, the figure shows that the point are further apart. Figures 5.6 and 5.7 refer to the SupCon-S method, a method whose results improved the most with the implementation of this solution. Figure 5.6 shows the plot of the SupCon-S method before the solution was applied. As in the previous examples, the points in this figure are quite close together, and the following can be highlighted:

(a) In the circle identified as `(a)`, there are two superclasses that are quite close, the superclasses `people` (in dark gray in the figure) and `large carnivores` (in purple in the figure);

(b) In the circle identified as `(b)`, the superclass `reptiles` also has two very close points;

(c) In the circle identified as `(c)`, the superclass `fish` (identified in pastel blue in the figure) has two very close points.

After the implementation of the solution, the evolution of the distribution of points can be seen in the figure 5.7. Unlike the previous examples, in which all the points were quite far

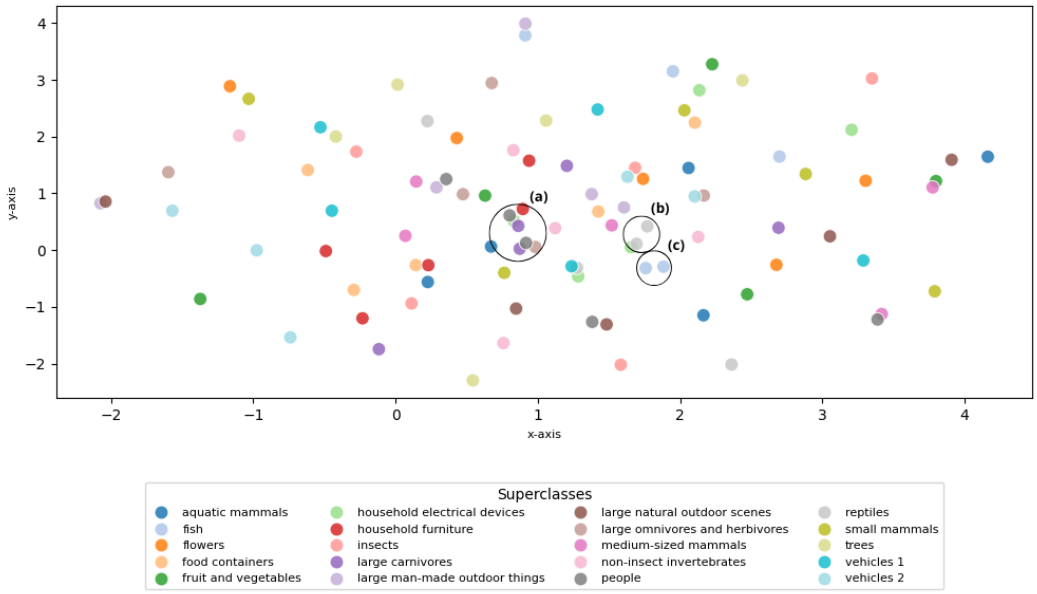

Figure 5.6: SupCon-S method before implementing this solution.

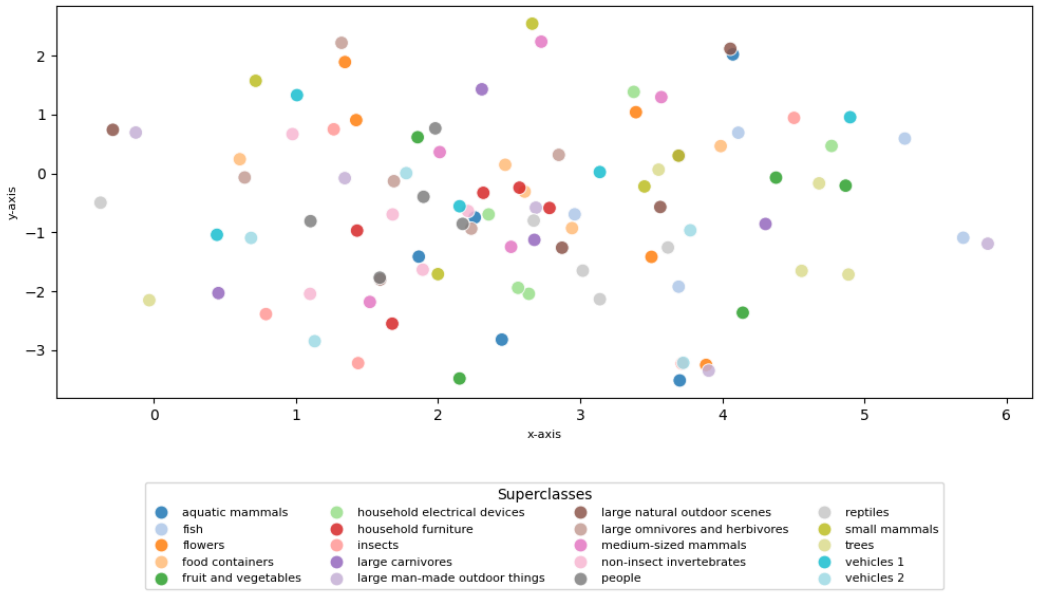

Figure 5.7: SupCon-S method after implementing this solution.

apart and there was no overlapping of points in the figure, this is not the case here. In fact, in the central part of the graph, it can be seen two pastel green dots (whose superclass is `household electrical devices`) which are very close together. It can therefore be concluded from looking at this figure, compared to the previous ones, that the fact that the dots are further apart on the graph does not mean that the results are necessarily better. Finally, figures 5.8 and 5.9 show the differences between not applying and applying this solution, respectively. Figure 5.8 shows the distribution of classes among their respective superclasses

before the solution was implemented. In this case, one can highlight:

(a) The circle identified as `(a)`, which again contains the superclass `reptiles`, which has two dots together. It should be noted that this superclass has been very close together repeatedly in all the methods;

(b) In the circle identified as (b), the superclass `household furniture` (in red in the graphic), has three joined dots.

Figure 5.9 shows the distribution of classes by superclass after applying the solution to the SupCon-M method. As in the previous case (please check figure 5.7 again), the after applying the change also shows that the points are not too far apart. It should also be noted that in both of these figures the scale is smaller than in the case of the two SelfCon methods.

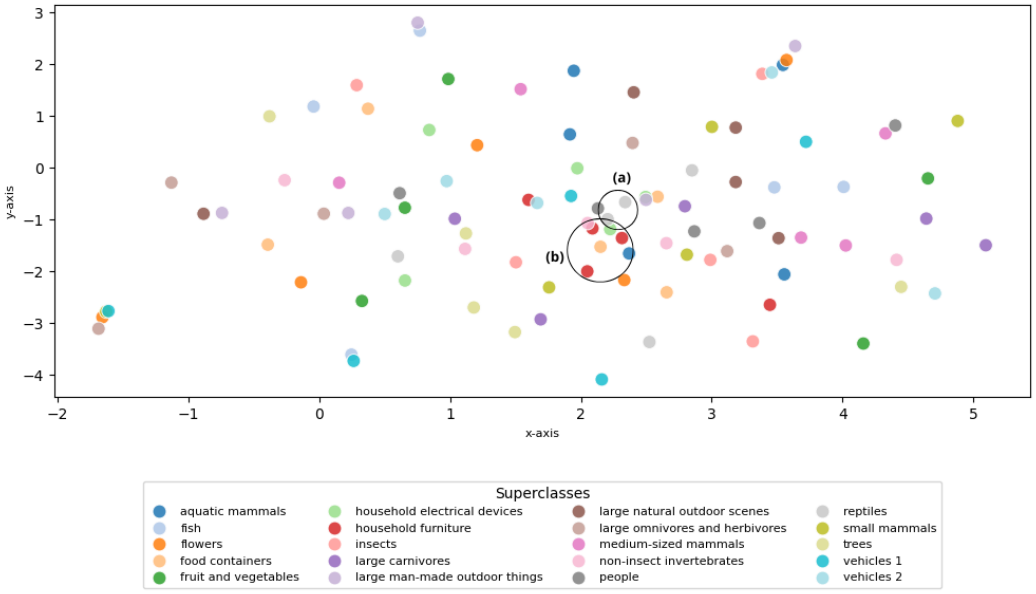

Figure 5.8: SupCon-M method before implementing this solution.

5.4 Final Considerations

Looking closely at all the solutions developed to overcome the limitations, some final considerations can be drawn:

- The different solutions made to the model worked better in single-view settings. In general, even if the results were lower than the base results, the results obtained by methods with a single-viewed framework were always better;

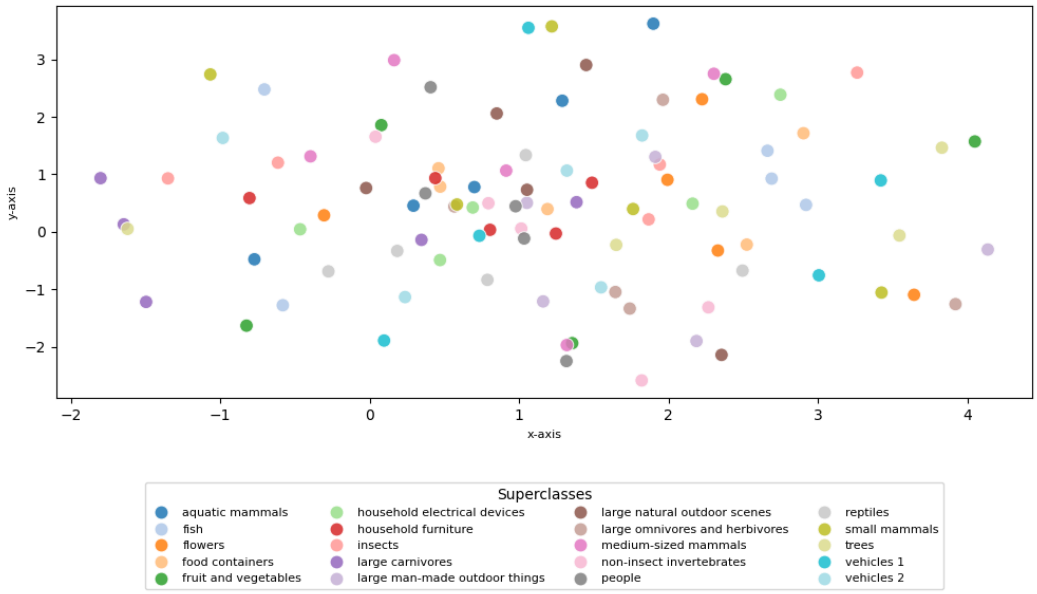

Figure 5.9: SupCon-M method after implementing this solution.

- Methods with a single-viewed framework cope better with the proposed solutions, as the results in these settings are better than methods with a multi-viewed framework;
- While the results of the SelfCon method could drop a little after the implementation of the different solutions, the results of the SupCon method always dropped more in comparison, which leads to the conclusion that the first method copes better with the changes introduced than the second;
- The SelfCon method, in both frameworks, has better results than the SupCon method;
- All of these statements are true for all of the solutions made, with the exception of the second change, whose results do not agree with the conclusions drawn above, and with the exception of the third solution, whose results have remained very similar to the originals.

5.5 Conclusion

This chapter shows the results obtained by the different solutions developed to overcome some limitations of the chosen Contrastive Learning method, SelfCon. The solutions developed are evaluated using Acc@1, where, although most of the results did not improve on the baseline results, these initial results were promising, and further refinements to the proposed solutions could enhance performance even further.
The next chapter presents the main conclusions of this dissertation, as well as future work.

Chapter 6

Conclusion and Future Work

This chapter presents the main conclusions, in section 6.1, and then presents the work to be done in the future, in section 6.2.

6.1 Conclusion

The study carried out throughout this dissertation, evaluating state-of-the-art Contrastive Learning methods, especially the method selected for modifications, the SelfCon method, has shown that these existing methods already have positive results, but there is always room for improvement.
The five modifications that were explored in this work had different levels of success, with some remaining fairly close to the initial results, and even one of the modifications made slightly improving the results (the fifth modification). This means that perhaps if small changes are made to the modifications already proposed (some of which have already been thought to improve the results obtained), the modifications could in fact improve the results obtained, thus contributing to the evolution of methods in this topic of Contrastive Learning. Of course, all these modifications have their limitations, and this is something that will continue to be explored in more detail in future work.

6.2 Future Work

As future work, several aspects can be done in the future to continue this research:

- It is important to see how the number of neighbors impacts the model, specifically in the first modification. Five neighbors were initially considered, but could this number already be too large and be damaging the predictions made? Is it still too small, so the results do not improve? It is important to understand the best value to consider, and also to take into account the computational cost required and the evaluation of the model;
- Although there is an approach that determines the number of neighbors dynamically, modification 3, this is a very restrictive approach, because the maximum number of

neighbors defined is often used. This is not ideal either, and may not help the results of the model. It is important to improve this method by defining a maximum number of different neighbors, or even changing the threshold defined dynamically. This goes hand in hand with the point above, since if an optimal number of neighbors is found, this value could be useful in this approach;

- All the proposed approaches should be reviewed, in order to try to understand why methods with multi-view settings always have lower results than single-viewed frameworks. This is an aspect that should be improved, perhaps by creating a flag that separates these two types of settings;
- All the approaches were made with cosine similarity as the similarity function to be considered. It would be interesting to see how another type of function impacted the results;
- It would also be interesting to increase the number of predictions made by the model, since these could influence the choice of the final class in the modifications where neighbors are considered.
- It can be also tested penalizing specific superclasses, to see if the overall results improve with a selective penalization of classes within a superclass to be separated. The superclass `reptiles` comes to mind, which has always been shown to be very close together in space, as seen before.

Bibliography

[1] J. Janai, F. Güney, A. Behl, A. Geiger *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, no. 1–3, pp. 1–308, 2020. 1

[2] X. Dong and M. L. Cappuccio, "Applications of computer vision in autonomous vehicles: Methods, challenges and future directions," *arXiv preprint arXiv:2311.09093*, 2023. 1

[3] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*. IEEE, 2018, pp. 67–74. 1

[4] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023. 1

[5] Akruti Acharya. (2023) Full guide to contrastive learning. Last Access: 15 january 2023. [Online]. Available: https://encord.com/blog/guide-to-contrastive-learning/ 1

[6] Lilian Weng. (2021) Contrastive representation learning. Last Access: 13 january 2023. [Online]. Available: https://lilianweng.github.io/posts/2021-05-31-contrastive/ 1

[7] P. H. Le-Khac, G. Healy, and A. F. Smeaton, "Contrastive representation learning: A framework and review," *Ieee Access*, vol. 8, pp. 193 907–193 934, 2020. 1

[8] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 1, 7

[9] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 1, 9

[10] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/6b180037abbebea991d8b1232f8a8ca9-Paper.pdf 1, 9, 11

[11] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *Advances in neural information processing systems*, 2020. 1, 21

[12] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, "Sampling matters in deep embedding learning," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 13

[13] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2, 14

[14] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1597–1607. [Online]. Available: https://proceedings.mlr.press/v119/chen20j.html 2, 15

[15] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," *Advances in neural information processing systems*, 2020. 2, 17

[16] Julianna Delua, IBM. (2021) Supervised vs. unsupervised learning: What's the difference? Last Access: 9 january 2023. [Online]. Available: https://www.ubi.pt/Ficheiros/Entidades/91363/codigo_integridade.pdf 5

[17] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015. 8

[18] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017. 9

[19] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International Workshop on Similarity-Based Pattern Recognition*, 2015, pp. 84–92. 9

[20] M. Gutmann and A. Hyvärinen, "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of

Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 297–304. [Online]. Available: https://proceedings.mlr.press/v9/gutmann10a.html 12

[21] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 12

[22] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018. 12

[23] C.-Y. Chuang, R. D. Hjelm, X. Wang, V. Vineet, N. Joshi, A. Torralba, S. Jegelka, and Y. Song, "Robust contrastive learning against noisy views," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2022, pp. 16 670–16 681. 13

[24] C.-H. Yeh, C.-Y. Hong, Y.-C. Hsu, T.-L. Liu, Y. Chen, and Y. LeCun, "Decoupled contrastive learning," in *European Conference on Computer Vision*. Springer, 2022, pp. 668–684. 13

[25] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," *arXiv preprint arXiv:1808.06670*, 2018. 14

[26] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *Advances in neural information processing systems*, 2020. 18

[27] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *International Conference on Machine Learning*. PMLR, 2021, pp. 12 310–12 320. 19

[28] J. Cui, Z. Zhong, Z. Tian, S. Liu, B. Yu, and J. Jia, "Generalized parametric contrastive learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. 22

[29] S. Bae, S. Kim, J. Ko, G. Lee, S. Noh, and S.-Y. Yun, "Self-contrastive learning: single-viewed supervised contrastive framework using sub-network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 1, 2023, pp. 197–205. 22, 42, 43, 44

[30] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1134–1141. 24

[31] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised state representation learning in atari," *arXiv preprint arXiv:1906.08226*, 2020. 24

[32] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," *Advances in neural information processing systems*, 2019. 24

[33] D. Gordon, K. Ehsani, D. Fox, and A. Farhadi, "Watching the world go by: Representation learning from unlabeled videos," *arXiv preprint arXiv:2003.07990*, 2020. 25

[34] R. D. Hjelm and P. Bachman, "Representation learning with video deep infomax," *arXiv preprint arXiv:2007.13278*, 2020. 25

[35] C. Zhuang, A. L. Zhai, and D. Yamins, "Local aggregation for unsupervised learning of visual embeddings," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6002–6012. 25

[36] J. Li, P. Zhou, C. Xiong, and S. C. Hoi, "Prototypical contrastive learning of unsupervised representations," *arXiv preprint arXiv:2005.04966*, 2020. 25

[37] T. Xiao, X. Wang, A. A. Efros, and T. Darrell, "What should not be contrastive in contrastive learning," *arXiv preprint arXiv:2008.05659*, 2020. 26

[38] X. Wang and G.-J. Qi, "Contrastive learning with stronger augmentations," *IEEE transactions on pattern analysis and machine intelligence*, 2022. 26

[39] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015. 41

[40] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009. 41

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 42