Universidade Estadual de Campinas
Instituto de Computação

Thalles Santos Silva

Self-Supervised Methods for Representation Learning of Visual Features

Métodos Auto-supervisionados para Aprendizagem de Representações Visuais

CAMPINAS
2022

Thalles Santos Silva

Self-Supervised Methods for Representation Learning of Visual Features

Métodos Auto-supervisionados para Aprendizagem de Representações Visuais

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/ Orientador: Prof. Dr. Gerberth Adín Ramírez Rivera

Este exemplar corresponde à versão final da Dissertação defendida por Thalles Santos Silva e orientada pelo Prof. Dr. Gerberth Adín Ramírez Rivera.

CAMPINAS
2022

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Silva, Thalles Santos, 1987-
Sa59s Self-supervised methods for representation learning of visual features / Thalles Santos Silva. – Campinas, SP : [s.n.], 2022.

Orientador: Gerberth Adín Ramírez Rivera.
Coorientador: Luiz Fernando Bittencourt.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Aprendizado de máquina. 2. Visão por computador. 3. Aprendizagem não-supervisionada (Aprendizado do computador). 4. Aprendizado auto-supervisionado (Aprendizado do computador). I. Ramírez Rivera, Adín, 1986-. II. Bittencourt, Luiz Fernando, 1981-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Métodos auto-supervisionados para aprendizagem de representações visuais
Palavras-chave em inglês:
Machine learning
Computer vision
Unsupervised learning (Machine learning)
Self-supervised learning (Machine learning)
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Gerberth Adín Ramírez Rivera [Orientador]
Hélio Pedrini
Eduardo Alves do Valle Júnior
Data de defesa: 08-03-2022
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0002-9821-2743
- Currículo Lattes do autor: http://lattes.cnpq.br/0552190698786053

Universidade Estadual de Campinas
Instituto de Computação

Thalles Santos Silva

Self-Supervised Methods for Representation Learning of Visual Features

Métodos Auto-supervisionados para Aprendizagem de Representações Visuais

Banca Examinadora:

- Gerberth Adín Ramírez Rivera
  IC/UNICAMP
- Prof. Dr. Hélio Pedrini
  IC/UNICAMP
- Prof. Dr. Eduardo Alves do Valle Júnior
  FEEC/UNICAMP

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 08 de março de 2022

Acknowledgments

I want to thank the State University of Campinas (UNICAMP) for accepting me into the Masters's degree program in Computer Science. When I started this journey, I was working full-time as a software engineer, and by that time, conciliating a full-time employee schedule with the responsibilities of being a regular graduate student was challenging. Luckily, UNICAMP allowed me to take courses at a slower pace, and when I finished the coursework requirements for the Master's degree program, UNICAMP and the Institute of Computing (IC) accepted me as a full-time student. The journey was a bit longer than usual, but it was worth it. I am grateful for it all.

I want to take this opportunity to thank everyone that directly or indirectly contributed to this moment. This includes all the professors and my fellow students that I had the pleasure to work with. I am sure that every conversation, collaboration, and exchange of ideas we had, made me a little more prepared to reach this position. I cannot forget my friends from outside academia and my girlfriend, Ariadny Ricci. The words of encouragement gave me the strength to go ahead, even in the most challenging times. Likewise, all of my appreciation to my supervisor, Prof. Adín Ramírez Rivera, whom I consider a dear friend. Professor Adín has been a fantastic supervisor throughout this time. From day one, he has pushed me toward being able to accomplish all of my goals. I am grateful for the countless video conversations during the COVID19 outbreak and look forward to continuing to learn from him.

Lastly, and most importantly, I dedicate this work to my family. Specially to my mother, Maria Soledade Oliveira Santos Silva, and my father, Oscarlino Pinheiro da Silva. The years of the COVID19 outbreak were tough for our family. We sadly lost my mother, who was the pillar of our home and whom I loved very much. Nevertheless, I am sure she is thrilled right now because this accomplishment would not be possible if not for her. I like to think that my mother was directly responsible for my love of computer science. Starting from the lessons on how to properly type on a keyboard (learned on typewriters) to the amazing computer she gifted me when I was only a teenager, and of course, followed by the persisting encouragement to get into a public university in Brazil, all of it living in a minuscule city in the countryside of Bahia the state of Brazil. It seems as though she planned it all.

I look forward to the new challenges in my academic life. If there is one thing I learned in this process, it is that I want to continue growing and learning from the best.

Resumo

Extrair informações semanticamente relevantes de um conjunto de dados de imagens não anotado é um dos mais complexos desafios da área de Visão Computacional. Nos últimos anos, métodos de aprendizado auto-supervisionados conseguiram reduzir drasticamente a grande lacuna outrora existente, entre representações obtidas de forma supervisionada e aquelas adquiridas de forma não supervisionada. Com o crescimento ininterrupto na captura e armazenamento de dados não estruturados, somando-se o alto custo associado à criação de conjuntos de dados anotados, aprender características a partir de dados não rotulados, possui o potencial de desbloquear uma série de problemas relevantes, com redução significativa de custos. Esta dissertação apresenta algoritmos de aprendizado de máquina capazes de extrair características de dados de imagens sem a necessidade de qualquer tipo de supervisão manual. Apresentamos um panorama geral do recente paradigma de aprendizado auto-supervisionado. Destacamos a recente evolução dos métodos, desde a otimização de tarefas denominadas "pretext" expandindo-se a um conjunto de métodos que utiliza funções de custo contendo componentes contrastivos e não contrastivos baseados em similaridade entre representações visuais.

Como principais contribuições, apresentamos dois novos algoritmos de aprendizado de representações visuais auto-supervisionados. Primeiramente, apresentamos Consistent Assignment for Representation Learning (CARL), um método auto-supervisionado, que combina os benefícios de algoritmos clássicos de agrupamento com técnicas de auto-supervisão. CARL aprende representações de forma não supervisionada. Nossos experimentos demonstraram que tais representações podem ser utilizadas como ponto de partida para o aprendizado de novas tarefas, de maneira mais eficiente no que diz respeito à quantidade de dados. Nossos resultados se comparam a métodos atuais de última geração em vários conjuntos de dados, incluindo CIFAR10, CIFAR100 e STL10. Apresentamos um estudo aprofundado para investigar os pontos fortes e fracos do nosso método inicial. Baseado em nossas descobertas, desenvolvemos um novo algoritmo que rivaliza métodos atuais de aprendizado de representações não supervisionadas no ImageNet. Nossos métodos aprendem representações a partir de dados completamente livres de rótulos explícitos. Além disso, nossos experimentos demonstraram que as representações obtidas por nossos algoritmos reduzem a necessidade de grandes conjuntos de dados anotados no processo de aprendizado de novas tarefas. Em outras palavras, as representações apreendidas por Consistent Assignment of Random Partition Sets (CARP), são compactas e transferíveis em várias tarefas de visão computacional.

Abstract

Learning semantically meaningful features from unlabeled data has been one of the most challenging problems in Computer Vision (CV). Recently, Self-Supervised Learning (SSL) methods have managed to drastically reduce the gap between supervised and unsupervised pre-trained representations to learn downstream tasks. With the non-stopping growth of unstructured data and the high costs associated with creating annotated datasets, learning representations from unlabeled data can unlock a series of problems with significant cost reductions. In this context, this dissertation presents a series of algorithms to perform unsupervised representation learning from images completely free of human annotations. We perform a comprehensive overview of the recently popular field of SSL and highlight the evolution of self-supervised methods, from derivation and optimization of pretext tasks to a robust framework based on similarity optimization using contrastive and non-contrastive loss functions.

As our main contribution, we present two novel algorithms for SSL of visual representations. First, we introduce Consistent Assignment for Representation Learning (CARL), a self-supervised method that combines the benefits of classic clustering algorithms, such as K-Means, with SSL methods based on similarity in the embeddings space. We demonstrate that representations learned by CARL, in an entirely label-free way, can be used to learn new downstream tasks in a data-efficient manner. CARL's pre-trained representations perform equally well or better than current state-of-the-art (SOTA) methods on various representation learning benchmarks, including CIFAR10/100 and STL10. We conducted a thorough study to investigate the strengths and weaknesses of CARL and, based on our findings, we developed a new algorithm capable of rivaling unsupervised representation learning methods on the ImageNet-1M dataset. Consistent Assignment of Random Partition Sets (CARP) learns representations using a novel randomized clustering approach. We introduce a new pretext task based on random set partitions of prototypes. Our experiments demonstrate that CARP's representations are compact and transferable across many vision tasks. Moreover, CARP's representations perform equally well against contemporary representation learning methods while offering benefits such as small batch sizes and smaller memory footprint.

List of Figures

List of Tables

List of Abbreviations and Acronyms

Contents

Chapter 1

Introduction

In the last two decades, the field of Artificial Intelligence (AI) has seen substantial development in research and industrial applications. Machine learning algorithms such as Logistic Regression and Naive Bayes can recognize patterns from features and solve problems that seemed otherwise impossible to be solved by hard-coding knowledge into expert systems. Despite their simplicity, these shallow learning algorithms can perform relatively complex tasks such as product recommendation or learn to distinguish between spam from non-spam emails.

More interesting, the performance of shallow learning algorithms massively depends on the representations they receive as input [1, 2]. For instance, if we decide to build a spam email detector using Naive Bayes, passing a large body of raw unstructured email data to a classifier will not help. Instead, we need to find a different way to represent the text before feeding it to the classifier. Notably, one commonly used text representation for this kind of application is the Bag-of-words (BoW) model [3]. The idea is to represent the text so that the importance of each word in the text is easily captured. Namely, the term frequency of each word, which represents the number of times a word appears in the text, is a popular text representation that can be used to train the spam filtering model.

We can see the power of good representations for training machine learning models by looking at Figure 1.1. In this example, we might be interested in using a machine learning model such as Logistic Regression to find a linear separation, a line in 2D, between the blue and green circles. However, it is straightforward to see that a model that learns linear boundaries will not succeed in such an example because there is no way to separate the two classes using a line in this current state of the data. Luckily, if we change the input representation so that instead of passing the raw data, we pass in the square magnitude of the values, we will see that the data landscape will change completely, and a linear separation between the two groups becomes evident in the feature space. Indeed, representations matter.

Nevertheless, it is not straightforward to know beforehand how to change the data representation in such a way so that linear separations in the feature space become evident. Different features often have different properties that may or may not be suited to solve a given task. Take the BoW term frequency features as an example. These features focus on the number of occurrences of a word in the text but discard information such as grammar and word order. For other Natural Language Processing (NLP) tasks, where the

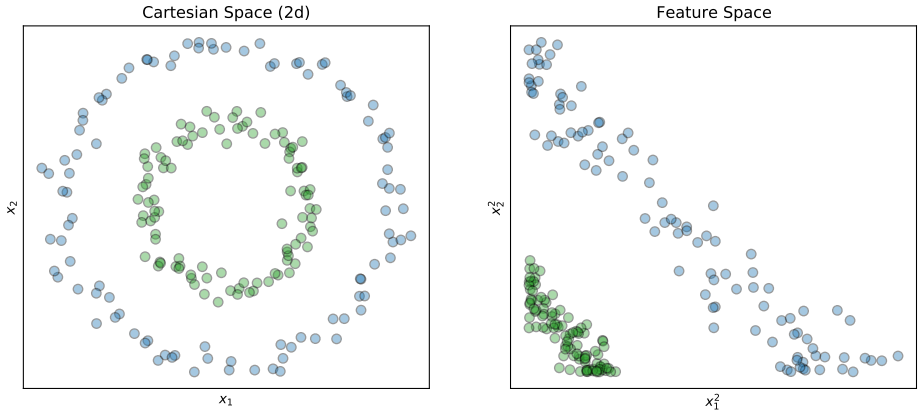

Figure 1.1. Representations matter for simple linear machine learning models such as Logistic Regression. A simple transformation, e.g., squaring the values of the raw features, might be enough to transform the feature space into linearly separable.

semantic relationship between words is necessary, the grammar and the order in which words appear in the text may be critical to solving that particular task. That is why the process of hand-designing features is considered such a challenging problem. For instance, imagine we want to build a car detector. We know that cars have some primary characteristics that make them different from other objects. Such components, one might argue, is the presence of wheels. In this scenario, to build car classifier, we need to have a way to represent the wheels of various types of cars. That is where the problem becomes complicated. After all, how could we create a wheel-detector that could generalize among all types of existing wheels and be robust to so many combinations of light, shape, and size distortions?

That is where Deep Neural Networks (DNNs) make a difference. With deep learning, we do not need to care about how to manually specify a wheel detector so that it can be robust to all types of existing wheels. Instead, by composing a series of linear and non-linear transformations in a hierarchical pattern, DNNs have the power to learn suitable representations by combining simple concepts to derive complex structures. In a nutshell, a classical supervised Computer Vision (CV) classifier would take raw data as input, and each layer iteratively refines the features from the previous layers. Thus, the neurons of the first layers learn fine-grained features such as edges and contours from the high-frequency input, and as the features traverse the hierarchy of layers, the subsequent neurons combine the fine-grained features to discover more complex concepts such as object parts. Lastly, we can use these refined representations and learn a linear classifier to map the inputs to a number of finite classes.

From this perspective, DNNs are representation learning models. At a high level, a typical supervised neural network has two components, (1) an encoder and (2) a linear classifier. The encoder transforms the input data and projects it to a different subspace. Then, the low-level representation (from the encoder) is passed to the linear classifier that draws linear boundaries separating the classes. Hence, instead of solving a task by mapping representations to targets (which a classical classifier would do), representation learning aims to map representations to other representations. These learned representations

are often dense, compact, and can generalize to similar data modalities. In other words, these representations can transfer well to other tasks and have been used to solve many other problems in which data annotations are hard or even impossible to get. Moreover, we can save hours of expert knowledge to create hand-designed features because learned representations usually offer much better performance and generalization properties.

In the current AI landscape, many industrial applications depend on supervised pre-trained models to solve different tasks for which annotated data is poorly available. The most common way is to take a large Convolutional Neural Networks (CNN) trained in a large annotated corpus such as the ImageNet [4], throw away its classification layer, and use the encoder as a feature extractor. In this way, we can map a high-dimensional input signal, such as an image, to a low-dimensional embedding vector that encodes that image's factors of variation. Then, we can use this new representation as input to shallow learning algorithms such as Support Vector Machines (SVMs). This recipe has been used with tremendous success in solving numerous industrial and research problems.

In this work, we focus on learning representations from images. However, we aim to develop techniques in which data annotations, often acquired by manual human labelers, are unnecessary. This activate area of research is referred to as deep unsupervised representations learning. Here, the word 'deep' hints that we use DNNs as the main building blocks of such learning systems.

Deep unsupervised representation learning seeks to learn a rich set of features from unlabeled data. The hope is that these representations will improve the performance of many downstream tasks and reduce the necessity of human annotations every time we seek to learn a new task. Specifically, in CV, unsupervised representation learning is one of the foremost, long-standing class of problems that pose significant challenges to many researchers worldwide. As computing power continues to rise and data continues to be ubiquitously captured through multiple sensors, the need to create algorithms capable of extracting richer patterns from raw data in a label-free way has never been so present.

In recent years, deep unsupervised representation learning has gained much attention because of the latest breakthroughs in NLP. The successes of Bidirectional Encoder Representations from Transformers (BERT) [5] and Generative Pre-Training Transformer (GPT) [6], both deep learning-based models that do not require manually annotated datasets in their training pipelines, have inspired a new generation of algorithms that made their way to CV applications. More specifically, the recently developed field of Self-Supervised Learning (SSL), which takes advantage of the supervision encoded in the data, is the main driver behind the latest successes in unsupervised representation learning for CV and NLP applications.

For example, BERT [5] is a deep unsupervised language representation model that learns contextual representations from unstructured text. Unlike context-free models such as word2vec or GloVe [7], which learns an embedding vector for each word independent of context, BERT learns representations based on the context in which each word appears. Moreover, BERT is optimized by solving a particular kind of SSL task that does not require manually annotated data. Namely, during training, a percentage of randomly chosen tokens are masked from the input sentence before going through the Transformer encoder. The encoder maps the input sentence to a series of representation vectors (one

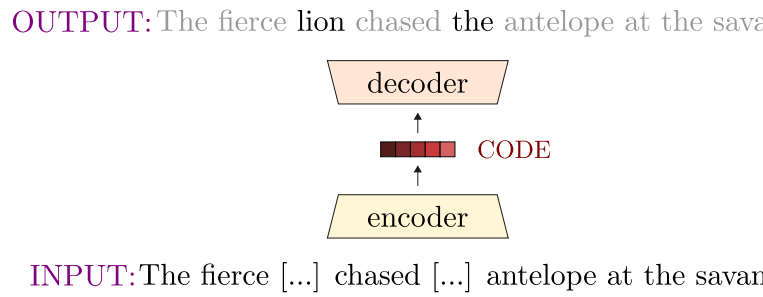

Figure 1.2. Masked Language Modeling MLM self-supervised pretraining objective used to train BERT. As input, the system receives a sentence where some of 'the tokens have been randomly removed. The network is trained to reconstruct the input signal, i.e., predict the missing words (tokens) given the available context.

for each word in the sentence). These hidden vectors are subsequently passed through a softmax layer that computes probabilities over the whole vocabulary so that the most likely words get a higher chance of being picked. In other words, it is a filling in the blank task where BERT aims to reconstruct the corrupted input signal from the partially available context. The decoder is trained to output the probabilities over all possible words so that the likely ones get a higher chance of being picked. This particular task is called Masked Language Modeling (MLM), and it has been used with success in BERT-based systems, including Liu et al. [8] and Lan et al.'s [9] work, refer to Figure 1.2.

Nevertheless, one very important characteristic of a system such as BERT is that, after training, we can fine-tune the BERT parameters on downstream tasks that do not have a lot of annotated training data. This practice usually achieves substantial accuracy improvements compared to training the system from scratch on downstream tasks. Moreover, this strategy reduces one of the main bottlenecks for training DNNs: data annotation.

Motivated by such examples, this dissertation presents a series of CV algorithms that learn general representations to ease the process of learning downstream tasks. We focus on the core methodologies used to develop state-of-the-art (SOTA) representation learning algorithms using DNNs [10].

These include:

- Self-supervised models capable of learning good representations by optimizing proxy tasks derived from the data [11, 12].
- Similarity-based methods such as the family of contrastive learning algorithms [13, 14] that seek to maximize the Mutual Information (MI) between samples [15].
- Deep unsupervised clustering techniques [16, 17] that look to categorize images by semantic structure.

This family of techniques has given substantial progress to the field of deep unsupervised representation learning. Recent work has attempted to combine these methods with relative success [18, 19]. Here, we explored the advantages and disadvantages of these learning algorithms and used them in conjunction to devise more effective unsupervised learning techniques.

We focused on two main approaches to unsupervised representation learning, (1) deep clustering and (2) SSL.

Over the years, clustering has been among the leading methods to learn the structure non-annotated data. Classic methods, e.g., K-Means, optimize a distance metric as a proxy to group similar images around a common centroid. Caron et al. [16, 17] have attempted to combine classic clustering with deep learning. Moreover, Li et al. [19] also suggest that combining clustering methods with self-supervised pretext tasks is a prominent technique.

On the other hand, SSL is an approach to unsupervised learning, and it is concerned with learning semantically meaningful features from unlabeled data. Approaches to SSL usually require a strategy in which a subtask is created out of unlabeled data in such a way that it can be optimized using traditional supervised learning techniques. This subtask, or pretext task, acts as a proxy that makes the network learn useful representations from the data. In other words, by solving the pretext task, the network learns good representations that can be used to train classifiers on different downstream tasks.

While classic unsupervised methods do not use any labels during training, most of the proposed pretext tasks use the same framework and loss functions as classic supervised learning algorithms. Therefore, self-supervised pretext tasks also require labels for optimization. However, the labels (or pseudo-labels) used to optimize these pretext tasks are derived from the data or its attributes alone.

In general, self-supervised pretext tasks consist of taking out some parts of the data and challenging the network to predict that missing part [20]. It can be predicting the next word in the sentence based on the previous context or predicting the next frame of a video based on the preceding ones. In fact, such pretext tasks have seen massive success in NLP applications, mainly because text data is discrete. Moreover, naively applying these same concepts to CV applications is hard because images are continuous, making the space of possible solutions much more complex.

Besides, creating self-supervised pretext tasks might resemble the process of hand-designing features for a classifier. It is not clear which pretext tasks work and why they work. Nevertheless, recent SSL methods have deviated to a general form of representing pretext tasks using a contrastive-based learning approach.

Contrastive SSL algorithms seek to learn representations by approximating similar concepts while pushing apart different ones. As of this writing, contrastive learning methods hold SOTA performance in unsupervised representation learning, making it an essential part of this project.

This dissertation presents novel approaches for unsupervised representation learning. We aim at combining the benefits of two popular strategies to unsupervised learning: SSL and deep clustering. We explore the collaborative nature of both techniques and design proxy tasks for learning generalizable visual representations that explore semantic structure in SSL tasks.

1.1 Motivation

Throughout recent years, supervised learning has been the leading and successful recipe for training DNNs. The availability of large, and most importantly, annotated datasets such as the ImageNet, allowed researchers to train DNNs that learns representations in terms of a hierarchy of concepts. These representations not only solve the task at hand, image classification in this case, but because of its generalization properties, they can also be used as good starting points to learn different downstream tasks such as object detection, segmentation, pose estimation, and more.

The recipe is simple, take a large annotated vision corpus, train a CNN model, then transfer its knowledge to solve a second task that usually does not have sufficient labels to train a deep network from scratch. This concept, called transfer learning, has been used successfully in many domains to solve numerous tasks and has been exhaustively used in commercial applications. Refer to Figure 1.3 for a graphical description.

The main limiting factor to scale this recipe is acquiring annotated data. To have an idea, the ImageNet dataset, which is a standard dataset among CV researchers, contains 14 million images with roughly 22 million concepts. However, if we compare ImageNet to the corpus of all available internet images, ImageNet is down by approximately five orders of magnitude [21].

Besides, if we look at the CV dataset landscape, the number of available annotated samples reduces drastically depending on the task at hand. Consider only the most popular CV tasks such as object classification, detection, and segmentation. In this scenario, as the level of prediction becomes more complex (from whole-image labels to pixel-level annotations), the number of labeled examples decreases substantially [22]. For this reason, most of the solutions to object detection [23–25] and segmentation [26–28] always take ImageNet pre-trained networks as starting points for optimization.

Moreover, ImageNet pre-trained models might not be the ideal solution for problems with a significant change in the domain. That might be the case for numerous applications in health care, such as classification and detection of breast cancer metastases in whole-slide images [29]. In such applications, the volume of curated annotated samples is minimal if compared to all available raw data. The process of annotating such records usually requires hours of specialized expert pathologists who need years of training to be capable of performing such a job—in situations like this, annotating a large dataset becomes prohibitively expensive and time-consuming.

Also, the rate at which humans annotate data cannot scale to the volume of the data we generate. Put it in another way, if we do not know the task to be optimized a priory, it might be very complicated and expensive to prepare a sizeable supervised dataset required by deep learning training.

All of these examples highlight the importance of learning generalizable representations from non-annotated data. Many research areas, including semi-supervised, weakly-supervised, and more recently, SSL, try to learn representations that can be transferred to new tasks using just a few or not using annotated examples at all.

Semi-supervised learning [30] aims to combine the benefits of having a few annotated observations with a much larger dataset of non-annotated examples. On the other hand,

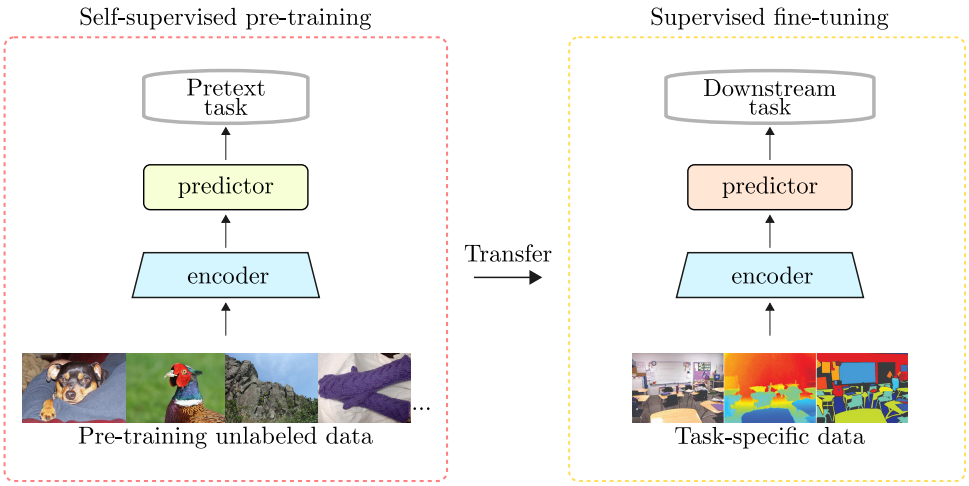

Figure 1.3. Self-supervised pre-training for computer vision. First, a DNN is trained by optimizing a pretext task in a self-supervised manner. Afterward, we can use the self-supervised encoder as a feature extractor and learn a classifier on a downstream task. Using the self-supervised encoder as a starting point reduces the requirements for large annotated datasets to learn downstream tasks.

weakly supervised learning [22] explores a large amount of noisy, and most of the time, imprecise labels as supervised signals. Finally, SSL [31], which is the main focus of this work, relates to devising proxy-tasks from which annotations derive from the data or its properties. Here, by optimizing pretext tasks, the network learns useful representations that can be transferred to optimize downstream tasks.

We can think of SSL as a pre-training step. First, a deep neural network encoder is optimized using a self-supervised objective function. This process occurs in a label-free way. The network learns useful representations by optimizing more general loss functions. Second, we use the pre-trained encoder to learn new downstream tasks. The pre-trained encoder facilitates downstream task learning by reducing the need for supervised data to achieve high accuracy. This process is summarized in Figure 1.3.

1.2 Contributions

This dissertation describes two novel algorithms for learning representations in a self-supervised label-free way for CV. The two approaches learn unsupervised representations that perform equally well or better than current SOTA methods in many unsupervised representation learning benchmarks. Unlike most contemporary work, our proposals combine ideas of clustering and self-supervision in novel ways.

The main contributions of this dissertation are:

- A framework that leverages current SSL methods with clustering-based algorithms for unsupervised representation learning. Consistent Assignment for Representation Learning (CARL) is based on two main ideas, consistency assignment of views to prototypes and trivial solution avoidance by penalizing non-uniform distributions of views over prototypes.

- A novel approach to avoid collapsed solutions in joint-embedding training architectures. Instead of relying upon additional pre-clustering steps using, e.g., K-Means

or non-differentiable algorithms to solve the clustering assignment problem, such as the Sinkhorn-Knopp [32], our method avoids collapsed solutions using an end-to-end online methodology with gradient descent.

- A method based on random partition sets that attenuate most of our previous method's limitations regarding batch sizing and training stability. Instead of enforcing consistency and uniform view assignment overall prototypes, we randomly partition the set of trainable prototypes into disjoint blocks during optimization. This approach improves the overall performance of the learned representations and increases the method's stability during training.

1.3 Outline

The rest of the dissertation proceeds as follows. We start by introducing background concepts in Chapter 2. We present a series of self-prediction tasks for SSL, the intuitive reasoning behind their success in learning unsupervised representations and describe some of the limitations of this learning approach. We continue with more introductory concepts in Section 2.3, where we briefly present the framework of Energy-Based Models (EBMs) which is going to be extremely useful to understand modern SSL methods. Next, in Chapter 3 we talk about contrastive and non-contrastive representation learning methods for CV. We present a standard set of key ideas used by current SSL methods, including our own. Then, in Chapter 4 we describe the existing methodologies to assess the quality of unsupervised representation learning algorithms and the list of datasets used in this dissertation to train and evaluate our methods. These evaluation protocols were used to compare our methods with existing algorithms. The first proposal is in Chapter 5. We introduce CARL, an unsupervised representation learning method that combines the benefits of SSL and deep clustering. CARL is based on two main ideas, (1) consistency assignment of views to prototypes and (2) trivial solution avoidance by penalizing non-uniform distributions of views over prototypes. Our second proposal is defined in Chapter 6. Here, we describe how we can outperform the results achieved by CARL, improving upon its main limitations by devising a novel approach based on set partitions to learn unsupervised representations. Lastly, in Chapter 7 we close this dissertation by pointing to future directions of SSL and describe how the methods presented in this work can be used to learn representations from different data modalities such as video or audio signals.

Chapter 2

Self-Supervised Learning of Visual Representations

This chapter presents a revision of Self-Supervised Learning (SSL) methods for representation learning of visual features. We start by addressing some of the early methods to devise and optimize Self-prediction Tasks (SPT) as a way to learn representations without labels. In the context of SSL, a SPT is an optimization task posed at the individual data point level. Usually, some part of the data is intentionally hidden or corrupted, and the network is challenged to predict that missing part or property of the data from the available information. Since the network only has access to some part of the data point, it needs to leverage intra-sample statistics to solve the task. These tasks are also known as pretext tasks, and they act as proxies to learn representations from unlabeled data. When optimized using such pretext tasks, a neural network can learn features that can generalize across different tasks and thus, ease the process of learning downstream tasks reducing costs, including computing time and data annotation.

2.1 Self-Supervised Learning

We can view SSL as a series of prediction tasks that aim to infer missing parts of the input from the partially visible context. In general, it relates to the idea of devising pretext tasks to predict an occluded portion of the data from the partially observed context. These concepts are prevalent in Natural Language Processing (NLP), where we can learn word-embeddings by predicting neighboring words based on the surrounding context [33].

Researchers have proposed many self-supervised tasks to learn representations from image data. For instance, generative-based pretext tasks attempt to learn the data manifold by posing a reconstruction task on the input space to recover the original input signal from a usually corrupted input. This category includes popular methods such as Variational Auto-Encoders (VAEs) and Generative Adversarial Networks (GANs).

Kingma and Welling [34] proposed a Bayesian approach to VAEs where the decoder generates data conditionally on a latent variable z while the encoder is trained to infer the latent variable from the input. In the GANs setup, Figure 2.1, the encoder takes in the latent variable z and learns to produce a data point $\tilde{x} = g(z)$ that comes from the original

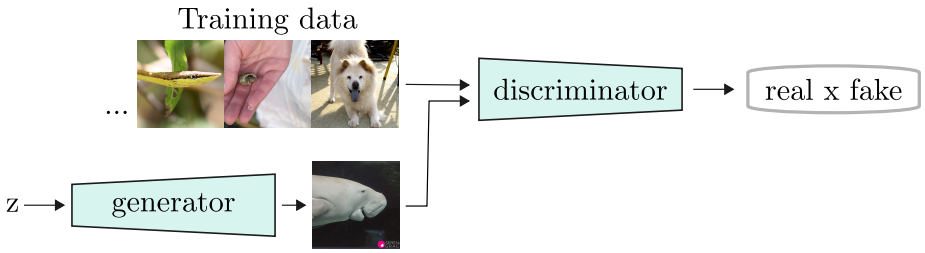

Figure 2.1. To train GANs, the generator function $g(\cdot)$ receives a latent variable z as input and attempts to produce a data sample $\tilde{x}$ that resembles an actual data point x. At the same time, the discriminator network is trained as a critic that needs to tell apart the sample created by the generator from the real sample from the training data.

data distribution, where the function $g(\cdot)$ is a neural network known as a generator.

Implementations such as BiGAN [35] trains an additional encoder function to invert the encoding process in order to map a data point x to a latent representation z.

Another class of generative models attempts to predict the values of a high dimensional signal, such as an image, in an autoregressive manner. These implementations make predictions at the pixel level. Using a raster scan order, they predict the next pixel value conditioned on the values they have seen before. Examples include PixelRNN [36], PixelCNN [37], and ImageGPT [38]. Although representation learning may not be the primary objective of such methods, they all have been used as feature extractors, in a transfer learning scenario, to learn downstream tasks.

Analogous to the task of masked language modeling in NLP, masked prediction tasks for images include denoising autoencoders and image inpainting. Vincent et al. [39] proposed to learn robust representations by learning denoising autoencoders. Here, a given input image is changed by a noise distribution that corrupts randomly chosen pixel values. The network receives the noisy input and rebuilds the original image using a reconstruction loss in the image space, Figure 2.2.

Similarly, Pathak et al. [40] present a generative model to learn visual features by intentionally masking a portion of the input image and challenging a deep neural network to fill in the missing regions. Besides the reconstruction loss, the system is optimized with an adversarial loss that further improves the reconstruction and the quality of the representations. Refer to Figure 2.2.

2.1.1 The Exemplar Pretext Task

The Exemplar-CNN proposed by Dosovitskiy et al. [11], is an example of a self-supervised method for learning transferable representations. Built on top of the concepts presented by Malisiewicz et al. [41], the main idea is to devise a prediction task that treats each image (an exemplar) and variations of it as a unique surrogate class. Each surrogate class contains randomly transformed variations of an image. These variations, also called views, are created by applying stochastic data transformations to the input image. Such transformations include cropping, scaling, color distortions, and rotations. Each set of views from the same exemplar gets assigned a unique surrogate class, and the network is optimized to correctly classify views from the same image as belonging to the same surrogate class. The exemplar pretext task is optimized using the cross-entropy loss and

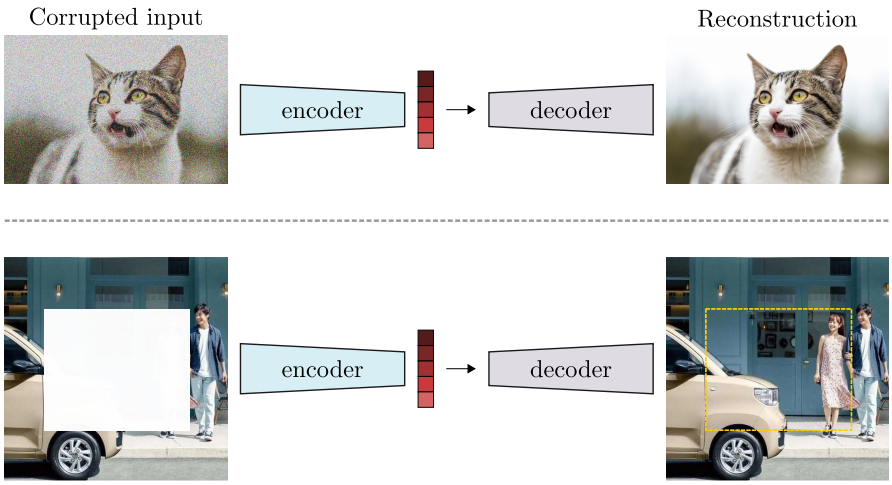

Figure 2.2. Pictorial representations a the denoising autoencoder (top) and a contextual autoencoder for image Inpainting (bottom). Both systems receive a corrupted image and attempt to reconstruct the original signal.

was trained with 32 000 surrogate classes. In its time of publication, the Exemplar-CNN outperformed the current state-of-the-art (SOTA) for unsupervised learning on several popular benchmarks, including STL-10 [42], CIFAR-10 [43], Caltech-101 [44], and Caltech-256 [45, 46].

Since there is a one-to-one correspondence between an image and a surrogate class, the exemplar pretext task becomes challenging to scale to enormous datasets such as the ImageNet with 1.3 million records. To address this limitation, Doersch and Zisserman [12] proposed to use the Triplet Loss [13] function as a way to scale the task to larger datasets. In short, the exemplar pretext task is a framework that aims to train a classifier to distinguish between different input samples. This strategy forces the network to learn representations that are invariant to the set of data augmentations used to create the views. Moreover, the concept of creating different views by using heavy data augmentation, now popularly referred to as instance discrimination, is one of the foundations of current SSL methods.

2.1.2 The Relative Patch Prediction Pretext Task

A contemporary work by Doersch et al. [47] proposed the relative patch prediction pretext task. This method introduces the idea of training a neural network to predict the relative positions of patches in an image. The learning framework is elegantly simple. First, a random patch is extracted from an image and used as the center relative position. From the center patch, we can extract 1 of 8 other patches (in a grid fashion) such that the relative position between the central patch and one of the surrounding patches is deterministic. Given a pair of patches as input, the network is trained to learn the relative position of the central patch with respect to its neighbor. Figure 2.3 depicts the training architecture.

The goal is to learn a visual representation vector (an embedding) for each patch, such that patches that share visual semantic similarities also have similar embeddings. The relative patch prediction pretext task is optimized as an 8-way classification problem

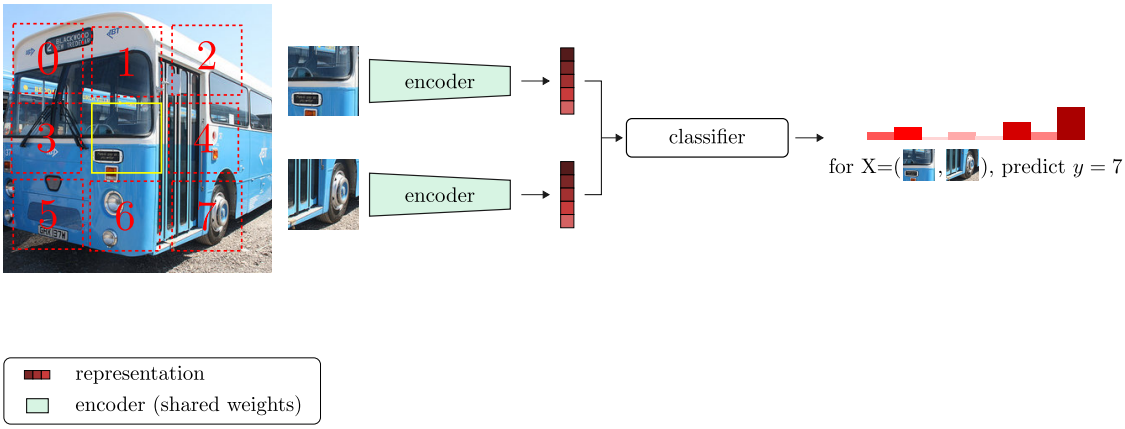

Figure 2.3. Overview of the relative patch prediction framework. The prediction task is designed as an 8-way classification problem where the network needs to predict the relative position of surrounding patches concerning the central patch.

using the cross-entropy loss. The feature representations learned through the relative patch pretext task achieved SOTA performance on the Pascal VOC 2011 [48] detection dataset. Moreover, different from the exemplar pretext task by Dosovitskiy et al. [11], the relative patch prediction task is much more scalable. For this reason, this work is considered one of the first successful large-scale SSL methods.

2.1.3 The Jigsaw Puzzle Pretext Task

Similar to the relative patch prediction pretext task, Noroozi and Favaro [49] proposed a pretext task to learn visual representations by solving jigsaw puzzles. From a single image, the formulation involves sampling nine crops (in a grid pattern) that are shuffled using a randomly chosen permutation p_i sampled from a set of permutations $\bar{P}$. The network receives the patches in random order, and the pretext task is to predict which permutation was used to rearrange the image patches. Since the number of possible permutations follows a factorial growth, the authors used a small subset of 1000 handpicked permutations. The subset of 1000 permutations is chosen based on their Hamming distances. Specifically, as the average Hamming distance between the permutations grows, the pretext task becomes harder to solve, and as a consequence, the network learns better representations. Intuitively, generating a maximal Hamming distance permutation set avoids cases where many permutations are very similar to one another, which would ease the challenge imposed by the pretext task. Fig 2.4 depicts the jigsaw puzzle Convolutional Neural Networks (CNN) architecture.

The jigsaw puzzle pretext task is formulated as a 1000-way classification task, optimized using the cross-entropy loss. Training classification and detection algorithms on top of the fixed representations leaned from the jigsaw pretext task showed advantages over the previous methods from Doersch et al. [47] and Pathak et al. [40].

2.1.4 The Rotation Prediction Pretext Task

Gidaris et al. [50] presented another simple and yet powerful self-supervised pretext task.

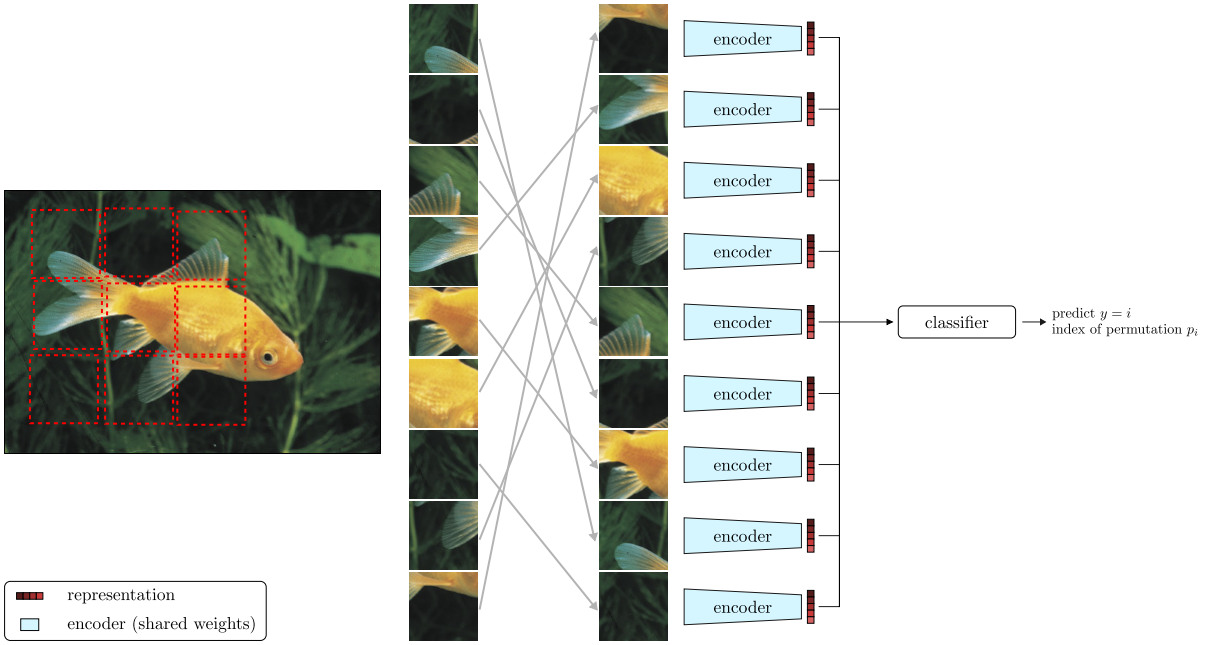

Figure 2.4. Learning representations by solving jigsaw puzzles. First, 9 crops from a grid-like structure are extracted and shuffled using a permutation p_i sampled from a set of permutations P. The crops are forwarded through a convolutional encoder using a Siamese architecture. The representations of each patch are concatenated and forwarded through a classifier that predicts the permutation used to shuffle the input patches.

The rotation prediction pretext task is a method to learn unsupervised visual representations by predicting which rotation angle was applied to an input image. For an input image x and a rotation angle β (randomly picked from a set of predefined values), the image x is rotated by an angle β and fed as input to a CNN. Then, the pretext task is to predict which of the valid rotation angles was used to transform the input image. The rotation prediction pretext task is designed as a 4-way classification problem with rotation angles taken from the set $\{0°, 90°, 180°, 270°\}$. The framework is depicted in Figure 2.5. Despite its simplicity, the rotation pretext task achieved SOTA performance on various unsupervised feature learning benchmarks, including classification on ImageNet [4], and transfer learning on PASCAL VOC [48].

2.2 Limitations of Pretext Task Optimization

One might ask why solving jigsaw puzzles or predicting rotation angles makes the network learn useful representations. Although the answer to this question is still open, helpful intuitions hint at why optimizing these pretext tasks results in learning useful visual semantic structures from data. To excel in a task such as relative patch prediction or jigsaw puzzle, the network needs to learn how different object parts relate to one another. Moreover, both tasks force the network to learn spatial context structure among the image patches. In other words, these tasks require the network to learn features that encode spatial reasoning.

Similarly, predicting random rotations drives the network to learn orientation concepts about objects. These concepts are well known and straightforward to humans but are very

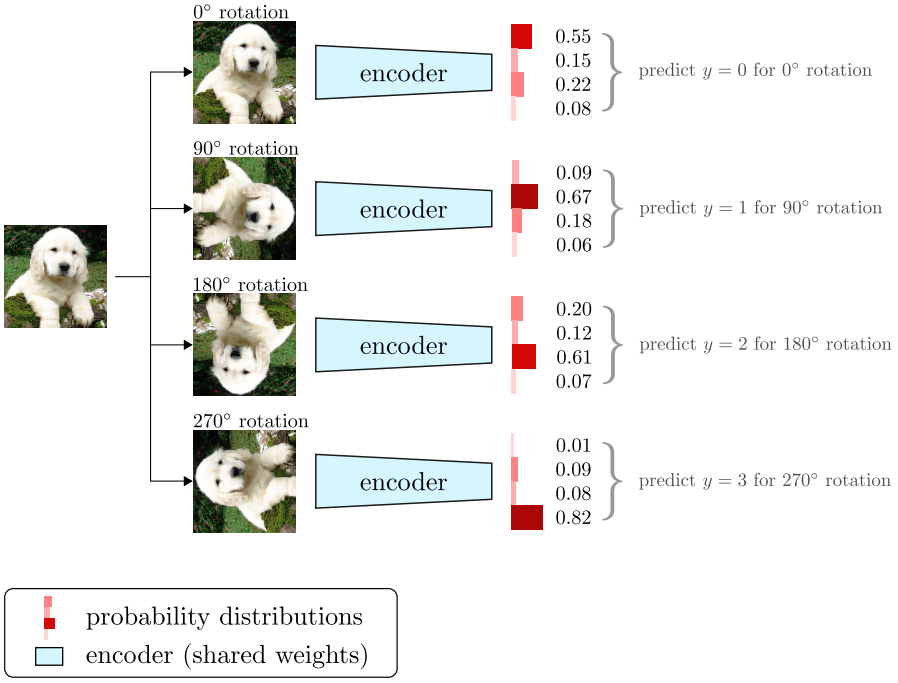

Figure 2.5. The rotation prediction pretext task learning framework. An image is rotated by a randomly picked angle β and then fed to a neural network. The task is designed as a 4-way classification problem in which the network must predict which rotation angle was used to transform the input image. Originally, the network was trained using a set of 4 rotation angles $\{0°, 90°, 180°, 270°\}$.

difficult to be encoded in computer systems. Such notions might include learning that trees grow upwards (instead of side-ways or downward) or that a person's head should be on top of the shoulders, not below them. In the same way, for contextual prediction tasks such as jigsaw and relative patch prediction, the network must learn spatial consistency features so that it can predict that a patch containing the nose of a living creature should go on top of the patch containing its mouth.

Another important characteristic of these pretext tasks is that they usually use the same loss functions of regular supervised training. In fact, from an optimization standpoint, the only substantial difference between these self-supervised methods and regular classification algorithms is the source of labels. While training a classifier often require human annotations, SSL extracts the supervisory signal from the data.

The idea of optimizing pretext tasks that extract training signals from the data has gained much traction in Computer Vision (CV). It has been applied to different domains, including video and audio-based applications [51–53]. Despite its potential, however, when comparing supervised and self-supervised-based representations for learning downstream tasks, early attempts to learn via self-prediction were still far behind.

Moreover, devising pretext tasks pose the disadvantage of creating the task itself. Indeed, creating such pretext tasks can be seen as a handcrafted procedure with no guarantees that optimizing for a given pretext task will push the network to learn semantically meaningful features that can transfer well to other tasks. Take the rotation prediction pretext task as an example. One would think that increasing the set of possible angles (from 4 to 8) would increase the quality of the final representations because the pretext task is now more challenging. However, an ablation study using multiples of 45-degree

rotation angles (from 0° to 315°) instead of multiples of 90-degrees showed that the overall feature performance actually decreased.

Another significant drawback of these manually created self-supervised pretext tasks is that optimizing them makes the network learn visual representations that covary with the choice of pretext task [54]. Moreover, Kolesnikov et al. [55] have shown that the quality of visual representations learned via pretext task optimization is very sensitive to the choice of CNN architecture. Put it differently, the choices of network architecture and pretext tasks seem to influence the quality of the final visual representation.

Driven by these findings, current methods [56, 57] for self-supervised visual representation learning seem to have deviated from optimizing pretext tasks to adopt a common framework based on contrastive learning algorithms [14] with join-embedding architectures. In essence, contrastive methods learn visual representations in the embedding space by approximating pairs of representations of the same concept.

This family of methods optimizes representations in the feature space and avoids the computation burden of reconstructing the input signal [58]. Moreover, recent contrastive and non-contrastive methods have achieved SOTA performance surpassing both self-supervised pretext tasks and latent variable models [35, 59] in nearly all representation learning benchmarks.

2.3 Energy-Based Models for Self-Supervised Learning

Thus far, we have seen many discriminative methods that allow Deep Neural Networks (DNNs) to learn representations by solving SPT in a label-free way. Many of these tasks propose a prediction problem at the image level. In other words, we devise a "supervised" task in which some part of the data or specific properties are held out and used as labels during optimization. These tasks are called "pretext tasks" or SPT because the labels come from the data, not from human labelers. Most of the pretext tasks we have seen so far were designed at the intra-sample level and did not consider inter-sample statistics. Before we head our attention to contrastive and non-contrastive SSL methods, here, we explore Energy-Based Models (EBMs) in the context of SSL. The framework of energy-minimization as an optimization task is a very intuitive tool to understand current SSL methods based on contrastive and non-contrastive loss functions. As we are going to see, the framework of EBMs can be used to derive many machine learning models, such as auto-encoder and GANs.

2.3.1 Energy-Based Models

EBMs [60] is a framework that can be used to derive many machine learning algorithms for supervised and SSL. EBMs are especially useful to model applications in which, for a single input, there are multiple reasonable outputs. Training an EBMs involves learning a scalar-valued energy function, denoted by $E(\cdot)$, representing the compatibility between variables. Given two variables x and y, the energy function $E(x, y)$ outputs a low value if y is compatible with x and a high value if y and x are incompatible. The variable x could be anything, from an image to text data or an audio signal. Similarly, y could be anything

that is supposed to be compatible with x, such as a list of class labels, a related image, a segmentation map, or an audio signal. In other words, EBMs pose the optimization task as learning dependencies between configurations of input variables through a scalar-valued energy function.

Training EBMs consists of finding the optimal set of trainable parameters that correctly shape the energy function. One common strategy is to feed the learning algorithm with input variables x and y that share semantics. It could be a couple of face pictures from the same person or two photos of the same chair taken from different angles. Then, the system needs to be optimized so that a particular x can only be compatible with observations that share similarities with it. In other words, while two face pictures from the same person must have low energy, two face pictures from different people should be incompatible and hence have high energy. To shape the energy function, the system needs to learn that for compatible pairs x and y, the energy must be zero, and for incompatible pairs, x and $\bar{y}$ the energy must be greater than zero, here, $\bar{y}$ represents an input variable that is non-compatible with x.

Once EBMs are trained, inference is the process of finding a value of y that is compatible with x. In other words, find values of y that make the energy $E(x, y)$ small. If $E(\cdot)$ is continuous and smooth, one option is to perform gradient-based optimization to find values of y. Since there might be multiple values of y, a different compatible value might be found depending on where the optimization starts at the energy surface.

2.3.2 Contrastive and Non-contrastive Energy-Based Models

In the context of SSL, there are two types of EBMs: (1) contrastive and (2) non-contrastive (regularized) methods. In contrastive methods, to shape the energy function, we need two opposing forces: (1) an attractive and (2) a repulsive (or contrastive) force. In this setup, we want to push down on the energy of compatible pairs, i.e. $E(x, y) = 0$, and push up the energy of non-compatible pairs, $E(x, \bar{y}) > 0$, for a non-compatible input $\bar{y}$. On the other hand, regularized methods are designed so that the volume of low energy is limited. As a result, by only attracting compatible pairs, i.e., minimizing the energy between x and y, we automatically maximize the energy of every other configuration of variables; in other words, the repulsive force is free.

These two types of EBMs can be used to interpret nearly all machine learning models. For instance, Denoising Auto-Encoders [39] and GANs [61] can be interpreted as contrastive EBMs.

For denoising auto-encoders, we want to learn a function that maps a data point $\tilde{x}$ from outside the data manifold (the ones we intentionally corrupt) to a data point x that sits on the manifold. Specifically, we have an encoder-decoder architecture in which the encoder receives the noisy input sample $\tilde{x}$, compresses it to a low dimensional representation z, and passes it to a decoder function that learns to undo the corruption on $\tilde{x}$ by forcing the reconstruction $\hat{y}$ to look like x. Thus, given an incompatible input sample $\tilde{x}$, we train a neural network to produce samples that are compatible with the observed data. This process assigns low energy between the reconstructed point $\hat{y}$ and the original data point x and assigns high energy between a point off the manifold $\tilde{x}$ and a point on the manifold

x, refer to Fig. 2.6. Note that the energy function here is the reconstruction error between the prediction $\hat{y}$ and the input image x.

Contrastive methods differ in the way they acquire negative samples for contrastive optimization. While some methods use complex heuristics to mine a set of relevant negatives, others abdicate the mining procedure and opt to pay the price of using a large number of negative samples.

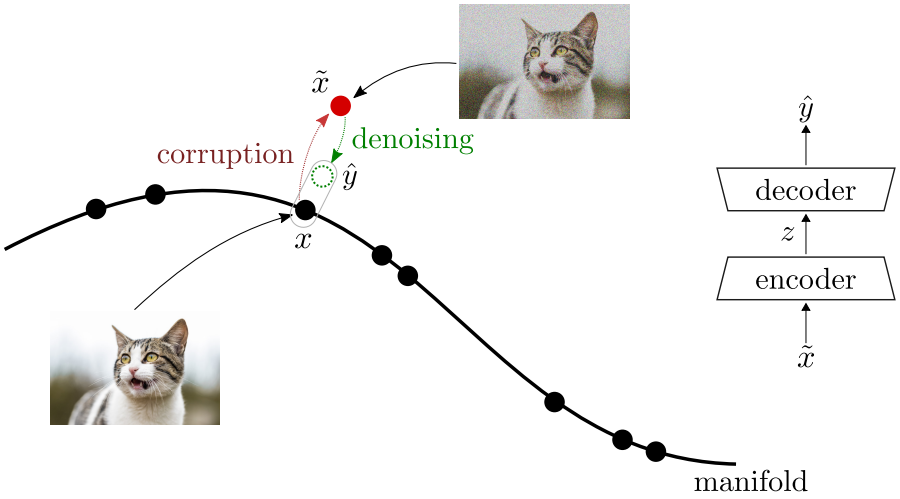

Figure 2.6. Denoising auto-encoders as energy-based models. A data point x is corrupted to $\tilde{x}$ so that it goes off the data manifold. The auto-encoder receives the corrupted observation as input and attempts to reconstruct it to $\hat{y}$. The system shapes the energy function by assigning low energy between the reconstructed point $\hat{y}$ and the input observation x and high energy between $\tilde{x}$ and x.

For GANs, we can interpret the discriminator as an energy function [62]. Here, we want the discriminator to assign low energy for an authentic image and high energy for a fake image that comes from the generator. At the same time, we train the generator to produce compatible images (has low energy) with the observations from the training set.

On the other hand, methods such as Principal Component Analysis (PCA), K-Means, and VAEs [34] can be interpreted as regularized EBMs. These methods vary in the way they limit the information capacity of the latent variables, but since the volume of low energy is bounded, only pushing down the energy of compatible pairs is enough to shape the energy function.

EBMs have the advantage of being very flexible in the way we pose the problem and perform optimization. Unlike probabilistic models, the energy is not normalized. Therefore, we do not need normalization terms that often involve dealing with non-tractable integrals to compute the partition function Z. Nonetheless, energy functions can be turned into probabilities by using the Gibbs Boltzman distribution.

Chapter 3

Contrastive Representation Learning

In Chapter 2, we described how one could take advantage of the supervisory signal hidden in the data to devise and optimize pretext tasks. These pretext tasks do not require manually annotated labels and optimizing such tasks give the network the capacity to learn generalizable representations that can be used to learn new downstream tasks. We also described some of the limitations regarding this algorithmic approach to learning representations. This chapter introduces a concept that merges the instance discrimination pretext task with joint-embedding architectures to learn unsupervised representations. As of this writing, contrastive and non-contrastive self-supervised methods are driving unsupervised representation learning to performance levels never seen before. Contrastive and non-contrastive learning methods can be very well understood through the lens of Energy-Based Models (EBMs), introduced in Section 2.3

In essence, we can describe a contrastive learning problem as follows: Given observations x^a, x^p, and x^n where x^a is an anchor, x^p a positive and x^n a negative, we want to learn a model to predict a high similarity score between x^a and x^p and to produce a low similarity score between x^a and x^n. That is,

$$\text{score}(x^a, x^p) > \text{score}(x^a, x^n). \tag{3.1}$$

In other words, the observations x^a and x^p share semantic meaning and the pair (x^a, x^p) represents a positive pair. On the other hand, the observation x^n is designed to be uncorrelated to x^a, and the pair (x^a, x^n) forms a negative pair.

3.1 Supervised Contrastive Learning

Contrastive learning ideas have been extensively used in machine learning. In supervised learning, for instance, it has been applied in a metric learning scenario where the ground truth labels are used to draw positive and negative examples. In metric learning, x^a and x^p are usually drawn as different images from the same class, such as two different images of a shepherd dog, while x^n is a random image from a different class, such as a picture of an airplane.

As an example, Goldberger et al. [63] have proposed a supervised non-parametric method for learning a distance measure to be used in the K-Nearest Neighbors (KNN)

algorithm. Neighborhood Component Analysis (NCA) learns a linear mapping of the input space after which classic KNN, using the Euclidian distance, performs well. Instead of choosing a fixed number K of neighbors and voting their classes, NCA proposes a randomized method where an input point x_i randomly selects a point x_j as its neighbor with probability p_{ij} given by the softmax of the distances, as

$$p_{ij} = \frac{\exp(-\left\|\mathbf{A}x_i - \mathbf{A}x_j\right\|^2)}{\sum_{k \neq i} \exp(-\left\|\mathbf{A}x_i - \mathbf{A}x_k\right\|^2)}, \tag{3.2}$$

where $\mathbf{A}$ is the learned projection matrix, and the summation is over all data points. The data point x_i inherits the class label from its selected neighbor x_j

The algorithm maximizes the expected number of data points indexed by i that will be assigned the correct class, for all points in the dataset as:

$$p_i = \sum_{j \in C_i} p_{ij}, \tag{3.3}$$

and the final objective,

$$L_{\text{NCA}} = \sum_i \sum_{j \in C_i} p_{ij} = \sum_i p_i, \tag{3.4}$$

is to learn a metric that maximizes the expected stochastic classification score, restricted to quadratic (Mahalanobis) distance measures.

NCA has some advantages over regular KNN for certain scenarios. It does not require the tuning of the neighborhood size K, and it does not make assumptions over the data, such as the shape of the class distributions or the decision surface.

Still in metric learning, Hadsell et al. [14] and Chopra et al. [64], proposed a similar method to learning a similarity metric discriminatively. These methods employ a very similar contrastive loss function that works on pairs of input images, defined by

$$L(\theta, y, x_i, x_j) = (1-y)\frac{1}{2}(D_\theta)^2 + (y)\frac{1}{2}\left\{\max(0, m - D_\theta)\right\}^2, \tag{3.5}$$

where the function $D_\theta(\cdot)$ is parameterized distance function to be learned as,

$$D_\theta(x_i, x_j) = \left\|g_\theta(x_i) - g_\theta(x_j)\right\|_2, \tag{3.6}$$

and g_θ is a neural network encoder.

The idea is simple, given a pair of training images x_i and x_j, and a binary label $y = \{0, 1\}$ indicating the level of compatibility between the input pair, if the pair of inputs is meant to correlate with each other, i.e., $y = 0$, then we want to minimize the distance between x_i and x_j, in the embedding space, using a distance function such as the Euclidian distance, Equation (3.6).

Otherwise, if the pair of inputs x_i and x_j is not considered similar to one another, i.e., $y = 1$, we want to make their representations as far away from each other by at least a margin defined by m, where $m > 0$.

If the distance between a negative pair (x_i, x_j) is smaller than the margin, they should

be pushed away from one another by at least the value of the margin. However, if the distance between a negative pair (x_i, x_j) is greater than or equal to the margin, it means that the negative x_j is already at an acceptable distance from x_i and it does not need to contribute to the loss function.

To learn such invariant mapping, g_θ is implemented as a siamese [65] Convolutional Neural Networks (CNN) with trainable parameters θ. In this setup, an input pair x_i and x_j are fed independently to the convolutional encoder g_θ, and the respective representations are used in Equation (3.5).

Contrastive training is particularly good for cases where the number of classes is large or not defined in advance. A CNN trained to recognize human faces from pictures will have access to a finite number of human faces for training. However, once deployed, the system must be able to perform inference and recognize faces that were not present in the training data. Moreover, the representations learned by g_θ have some desired properties, such as being compact and invariant to various geometrical transformations and lighting conditions.

Different from the pair-wise contrastive loss, Equation (3.5), where we either sample a positive pair or a negative pair, the triplet loss [13, 66], defined as

$$L = \sum_{i}^{N} \max(0, D_{pos} - D_{neg} + m), \tag{3.7}$$

requires at least three examples, an anchor, a positive, and a negative. The function $D(\cdot)$ is a normalized squared l2-distance, as

$$D_{pos} = \|g_\theta(x_i^a) - g_\theta(x_i^p)\|_2^2 \tag{3.8}$$

$$D_{neg} = \|g_\theta(x_i^a) - g_\theta(x_i^n)\|_2^2, \tag{3.9}$$

where g_θ is a CNN that encodes a high-dimensional image signal into a low dimensional representation vector.

Given an anchor image x^a and a positive x^p, the pair (x^a, x^p) forms a positive pair. Similarly, an anchor and a negative example x^n form a negative pair (x^a, x^n). The anchor works as a relative point. The goal is twofold, we want to minimize the distance from the positive to the anchor and to maximize the distance from the negative to the anchor. The margin m adds a constraint such that the distance between the negative point x^n and the anchor should be at least m larger than the distance between the positive x^p and the anchor.

Hard negative mining is an essential operation for optimizing the triplet loss. Ideally, we want to choose the most offending negatives (the ones within the red circle), which are the ones closer to the anchor, Figure 3.1. This strategy is important because picking easy negatives (the ones in the green square) makes the problem easy to solve (trivial) and slows convergence.

Similarly, the triplet loss in Equation (3.9) is optimized using a siamese network architecture. Here, each of the triplet images (anchor, positive, and negative) is independently passed through a convolutional encoder f_θ, and their respective representations are used in the loss, Equation (3.9).

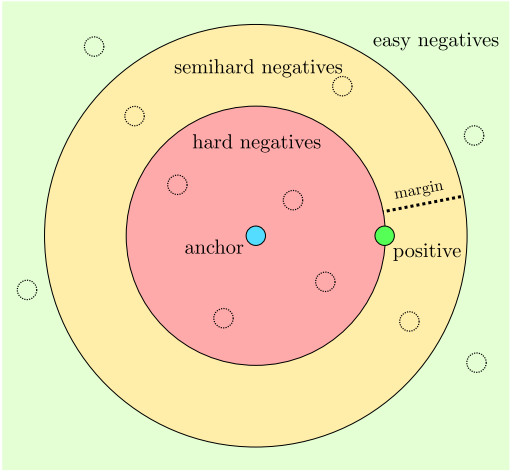

Figure 3.1. The triplet loss requires an anchor, a positive, and a negative example. We want the positive to be as close as possible to the anchor and the negative to be as far away from the anchor as possible. Negatives within the red circle contribute more to training convergence, while negatives in the green region may produce trivial solutions.

The contrastive pair-wise loss, Equation (3.5), and the triplet loss, Equation (3.9), both optimize over a single negative example. Now, consider the contrastive loss in Equation (3.5). It works on input pairs. If the two input images come from the same class, a learning update brings the corresponding embeddings closer. If the pair of input images are from distinct classes, the loss optimizes the neural network's weights so that the embeddings for the anchor and negative are far from each other.

The triplet loss shares some of the same principles. Different from the contrastive loss, it works on inputs composed of triplets. The loss optimizes the trainable weights so that the distance between embeddings from the same class (anchor and positive) is smaller than the distance between embeddings coming from separate classes (anchor and negative).

In both cases, when we push representations to be far apart, we only do it between an anchor and a negative from a single class. This highlights the importance of negative mining. If we only get to contrast an anchor to a single observation from a different class, we would wish to pick the most challenging observation from that class so that our update gets a stronger signal. Otherwise, if we only pick moderately to easy negatives, convergence will take much longer, and poor local minima is a constant threat.

Ideally, we want to simultaneously push the anchor's representation far away from all other classes for every training update. Following this idea, Sohn [67] proposed an $(N+1)$-tuplet loss called multi-class N-pair loss (N-pair-mc loss). The N-pair-mc loss extends the triplet loss, Equation (3.9), by performing comparisons among many negatives instead of just one. The objective is to identify a positive observation among N-1 negatives.

From an input pair (x_i^a, x_i^p), and an encoder function g_θ that converts inputs into low dimensional representation vectors, the loss function is defined as

$$L_{\text{N-pair-mc}}\left(\{x_i^a, x_i^p\}_{i=1}^N ; g\right) = \frac{1}{N}\sum_{i=1}^{N} \log\left[1 + \sum_{j\neq i} \exp(g(x_i^a)^T g(x_j^p) - g(x_i^a)^T g(x_i^p))\right]. \tag{3.10}$$

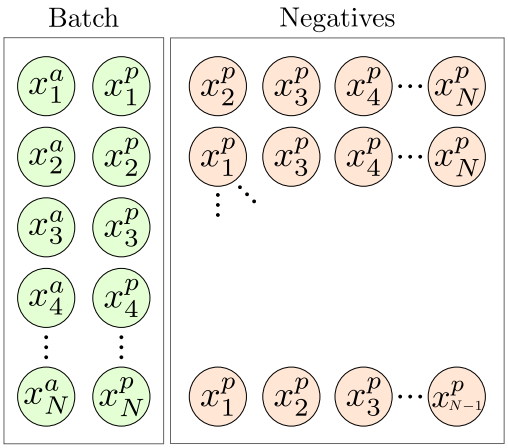

Figure 3.2. The N-pair-mc loss requires a batch containing N pairs of images. Each pair contains an anchor and a positive observation. These are two different images from the same class, and each pair within the batch represents a different class. For training, every pair is accompanied by an additional N-1 observations from the other pairs within the batch to be used as negatives.

Note how a given anchor is put against many negative exemplars instead of just one. For training, each batch contains N pairs $\{(x_i^a, x_i^p)\}_{i=1}^N$. Each pair contains two distinct images from the same class, i.e., an anchor and a positive image. Since each pair in the batch is from a different class, the representations from the other pairs in the same batch can be used as negatives in the loss function, Figure 3.2 depicts this process.

In a single step, the N-pair-mc loss approximates the embeddings of anchor and a positive in the feature space while pushing the anchor away from N-1 negative representations.

3.2 A Framework for Self-Supervised Learning

Unlike metric learning, Self-Supervised Learning (SSL) is interested in learning generalizable representations from unlabeled data. Given this constraint, it is not possible to guarantee the sampling consistency of positives and negatives with reasonable confidence. Put it differently, we cannot confidently draw a positive observation x^p from the same class as the anchor, nor can we sample a negative observation x^n that comes from a different class from the anchor. For this reason, recent contrastive and non-contrastive methods in SSL rely on a well-defined framework composed of three ideas.

3.2.1 Synthetic View Generation

Data augmentation is one of the critical ingredients for the success of SSL in Computer Vision (CV). These stochastic operations are primarily used to mimic different views of a concept. The process of view generation is akin to the exemplar pretext task of Dosovitskiy et al. [11]. It describes the idea of transforming an input image using random geometric and pixel intensity transformation, Figure 5.4. The goal is to transform an input image such that the semantic information is preserved, i.e., the class information remains unchanged, but the overall look and feel of the image might suffer radical modifications.

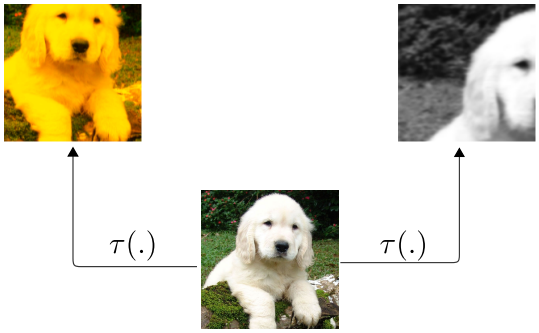

Figure 3.3. Synthetic view generation is one of the key ideas behind SSL. Since there are no labels to draw consistent positives and negatives, the workaround is to mimic different views of objects and entities in an image using random transformations. Here, an image of a dog is transformed by a function $\tau(\cdot)$ such that the class information remains unchanged while the image style is modified.

In other words, random transformations are designed to introduce non-trivial variations between the views constraint to preserve the image's semantic information. Since we know that the two transformed images come from the same source image, we know that they still represent the same concept despite visual differences. Hence, we can use these views as an anchor and positive because even though they look different, we are sure they share the same class information. The model, in turn, learns to represent the concept that does not change between the views.

Basic augmentation techniques for self-supervised pre-training include:

- Random crop and resize
- Color distortion
- Gaussian blur
- Color jittering
- Random flip/rotation
- Random grayscaling

Despite the success, using such transformation to create synthetic views also hint about the limitations and hidden assumptions of current SSL models. For example, consider a pair of views generated by the composition of the data transformations listed above. During self-supervised pre-training, a joint-embedding architecture is independently fed with the two views and produces corresponding embeddings optimized to be similar in the feature space. However, the data transformations used to create these views are largely based on successfully applied techniques to solve classification tasks. Moreover, to solve a classification task, the neural network does not need to care about some properties of the data, such as the object's exact location. In fact, data augmentations such as cropping and shifting tend to corrupt the image such that the location information is lost. As a result, if these augmentations are used to learn embeddings, the network will learn representations that discard information such as the object's position in the scene. Therefore, if we use these representations to learn downstream tasks that require the object's feature position,

such as detection and segmentation, the learned representations might not produce the expected result.

Recent self-supervised methods learn representations by treating augmented versions of the same record as instances of the same class [56, 57]. It is based on the assumption that good representations must be invariant to a number of transformations. In this context, SSL methods present a general framework to solve multiple discriminative tasks, in the embedding space, without reconstructing the original signal. However, a problem of current data augmentation techniques for SSL is that they are ad-hoc and specially designed for specific data modalities.

3.2.2 The Joint-Embedding Architecture

Another important idea behind recent self-supervised methods is the joint-embedding architecture, Figure 3.4. In this learning framework, two neural networks $f(\cdot)$ and $g(\cdot)$ are trained to produce similar embeddings for compatible inputs x and y. The compatible inputs x and y can be two images of the same class, such as two images of a dog, an image and a text description of the scene, or a sequence of video frames.

The siamese architecture Bromley et al. [65], where both networks share the same set of weights, is a particular case of the joint-embedding architecture.

Training join-embedding architectures pose the problem of avoiding collapsed solutions. In such situations, the networks learn to disregard the input signals and minimize the cost function by producing constant embeddings for all input signals.

In SSL, two views, derived from the same image, are fed to the encoders $f(\cdot)$ and $g(\cdot)$. For each view, the encoder produces respective representation vectors used in a cost function. The two networks $f(\cdot)$ and $g(\cdot)$ might be different functions, usually applicable when dealing with different modalities, or they might share weights.

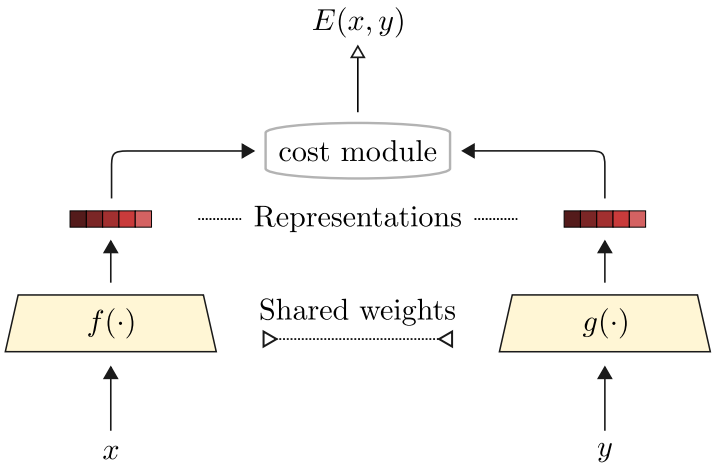

Figure 3.4. The siamese network is a particular instance of the joint embedding architecture. It processes a pair of input signals independently and produces corresponding representation vectors. The representations are fed to a cost function of choice, and the resulting scalar may be interpreted as an energy value.

3.2.3 The Cost Function

The loss function of such methods usually produces a scalar value that can be interpreted as an energy value. The goal is to shape the energy function such that representations from observations that share semantic information (images from the same class) are close to one another, while representations from images containing uncorrelated objects should be far apart. The loss function might have an explicit contrastive term, e.g., InfoNCE [57], or be regularized and not enforce a contrastive term explicitly.

Note that, instead of optimizing a pixel-level loss function such as the Mean Squared Error (MSE), which is typically used by reconstruction-based methods such as [20], self-supervised methods operate in the feature space. This characteristic offers significant advantages. To begin, since reconstruction is not enforced during optimization, there is no need to learn a decoder function, which significantly reduces the computational overhead and memory footprint of training. Moreover, optimization in the feature space allows the network to throw away information that could be useful for reconstruction but may not contribute to learning downstream tasks. In other words, the representations' properties that help discriminate between classes are not necessarily the same ones required to reconstruct the input signal.

As will be apparent, the main difference among methods that utilize this learning framework lies in the techniques they use to prevent collapsed solutions that might occur when optimizing join-embedding architectures. Indeed, an undesired trivial solution can be easily accomplished if the trainable weights are changed such that the network outputs constant representation vectors for all training data points.

3.3 Hard Negative Mining for Contrastive Learning

Hard negative mining is very important for contrastive learning. Given two images x and x^n, from different classes, image x^n is a hard negative for x if the embedding representation of x^n is very close to the embedding representation of x, even though they do not share class semantics. The issue is that we also expect an image x^p to have an embedding representation closer to x if x^p is a different view of the concept represented by x. For example, if x is a picture of a car and x^p is a picture of the same car under different lighting and angle configurations, x^p and x^n would similarly be placed closer to x. To avoid such situations, contrasting challenging negatives during training encourages the model to learn representations that distinguish hard-negatives from true positives.

We can extract negatives in two different ways: explicitly and implicitly. Explicit hard negative mining is very popular on Natural Language Processing (NLP) applications. In CV, MoCHi Kalantidis et al. [68] proposes a hard negative mixing strategy at the feature level that mines hard negatives by sorting the image embeddings according to similarity to the query. Robinson et al. [69] proposed to mine negatives by learning a distribution over hard negative pairs.

However, current SSL methods using contrastive learning rely on an implicit way of performing hard negative sampling. These methods take advantage of the in-batch samples and require very large batch sizes to maximize the probability of having true

hard negatives within the batch. Due to the large memory requirements of having large batch sizes, one strategy is to employ additional data structures such as large memory banks [56, 70]. The idea is to store a large number of feature representations from past observations and use them as negatives.

3.4 Contrastive Self-Supervised Learning

Contrastive Predictive Coding (CPC), proposed by Oord et al. [71], was very influential for the resurgence of contrastive methods for self-supervised representation learning. CPC is a learning framework that generalizes the notions of word2vec [72] to different domains, including CV and Audio signal processing. Given a high dimensional sequence of observations x_t, e.g., an audio signal or a sequence of patches extracted from an image, CPC uses a non-linear encoder $g_{enc}(\cdot)$ to map the sequence of input observations x_t to a corresponding sequence of latent representation vectors $z_t = g_{enc}(x_t)$. Next, it uses an autoregressive model $g_{ae}(\cdot)$ to summarize the representations z_t, in the latent space, to produce a contextual representation $c_t = g_{ae}(\mathbf{z})$ from all z_t. CPC takes the context vector c_t, along with a latent representation from a future time step z_{t+k}, and optimizes the system to maximize the mutual information between this pair of representations.

The CPC framework was applied to different data modalities, including speech, computer vision [73], NLP applications, and reinforcement learning. In all cases, the general approach is to predict a future signal from a past contextual vector. To do that, CPC proposes a loss function based on Noise-Contrastive Estimation (NCE) termed InfoNCE. The idea is to frame the problem as a discriminative task where the network needs to correctly identify pairs of positive samples among a large number of negative pairs. The loss is defined as

$$L_{\text{InfoNCE}} = -\mathbb{E}_X \left[\log \frac{s(x_{t+k}, c_t)}{\sum_{x_j \in X} s(x_j, c_t)} \right], \tag{3.11}$$

where the function $s(\cdot)$ is a score function defined as $s(x_{t+k}, c_t) = \exp\left(z_{t+k}^T c_t\right)$. In the numerator, the pair of representations (z_{t+k}, c_t) form the positive pair. Here, the representation vector z_{t+k} encodes a plausible future signal for the sequence encoded in the context vector c_t. On the other hand, in the denominator, negative representations z_j form negative pairs (z_j, c_t) that will have their representations pushed apart in the feature space. We can interpret the information encoded on the negative representations z_j as non-plausible futures for the context vector c_t. The problem is designed as an N-way softmax classifier with one positive and $N-1$ negatives.

Optimizing the InfoNCE loss, Equation (3.11), is equivalent to maximizing a lower bound on the Mutual Information (MI) between the representations that form the positive pair [71]. In this case, we aim to maximize the MI between the context vector c_t and the future target representations z_{t+k}. However, in practice, such optimization requires a large number of opposing samples.

For audio data, CPC is designed to predict a future audio section z_{t+k} from a past context vector c_t. Thus, the future audio chunk is treated as the target, and we want to learn a representation that can predict the actual future signal when contrasted with al-

ternative representations. To sample the negatives z_j, audio chunks from different records or sections of the same audio from different timestamps can be used.

Depending on how the negatives are sampled, the representations may learn different semantics. For instance, if the negatives come from audio signals of different speakers, the leaned representations might encode properties that can allow a more straightforward classification of the speaker. In other words, if we know the downstream task in advance, we can tweak the framework to learn representations with useful properties to solve the task at hand.

The InfoNCE loss, Equation (3.11), is similar to the triplet loss, Equation (3.9), from metric learning. However, instead of working with triplets, InfoNCE uses multiple negative samples. As discussed is Section 3.3, to have many negatives allows the system to perform hard-negative mining implicitly. In the InfoNCE loss, a hard negative among the N negatives will contribute much more to the final loss, while the easy ones will contribute very little.

Following similar ideas, Tian et al. [74] proposed a contrastive SSL method to learn unsupervised representations from multiple views. Contrastive Multiview Coding (CMC) builds on top of MI maximization [15, 71]. The main idea is to learn representations that model invariant factors across different views of an image. CMC empirically showed that using multiple views to create positive pairs contributes towards learning better representations.

Intuitively, having multiple positive views of the same concept allows the network to capture more information, which translates to powerful representations.

For ImageNet training, a source image is converted to the Lab image color space. The L and ab channels are split such that the channels L and ab, from the same image, form the positive pair. Similarly, ab channels from randomly selected images are paired with the anchor to form negative pairs. In the case of four positive views, the transformations include luminance (L channel), chrominance (ab channels), depth, and semantic labels.

The positive pairs are created by considering all possible combinations of pairs of views (i, j) such that $i \neq j$. Instead of using the N+1-way softmax classification with N negatives, CMC proposes an NCE approximation of the InfoNCE loss, Equation (3.11). According to empirical tests, the NCE approximation performed better when given the same number of negative observations. Moreover, to sample a large number of negatives without the need of recomputing feature vectors, CMC maintains a memory bank that stores latent features. Finally, unsupervised representations learned by CMC on ImageNet beat previous state-of-the-art (SOTA) methods like RotNet [50] and DeepCluster [16].

Around the same time, He et al. [56] followed by Chen et al. [75] presented Momentum Contrast for Unsupervised Visual Representation Learning (MoCo). Similar to previous work, MoCo uses the InfoNCE contrastive loss, Equation (3.11), combined with the instance discrimination pretext task. Unlike CPC [71], MoCo does not try to learn temporal coherent representations by modeling a context embedding and predicting a future time step. Instead, it uses the instance discrimination pretext task in which heavy data augmentation is used to synthesize positive views.

The instance discrimination pretext task follows the same ideas from the Exemplar-CNN from Dosovitskiy et al. [11], Wu et al. [70]. The purpose is to treat every possible

transformation of a given image as belonging to the same class. In other words, we want to treat augmented versions of the same image as different views of the same concept and, hence, express the same meaning from different perspectives. In this way, the network learns representations that map different views of the same concept to closing neighborhoods in the feature space, while images containing different concepts have their representations pushed apart.

MoCo uses two ResNet-50 [76] denoted query $f_q(\cdot)$ and momentum $f_m(\cdot)$ encoders during training. At each iteration, from an image x_i, two views are created $v_i = T(x_i)$ and $v_j = T(x_i)$ by applying a function $T(\cdot)$ containing random augmentations. Each network receives one of these views as input and produces respective embedding vectors as output, $z_i = f_q(v_i)$ and $z_j = f_m(v_j)$. The pair of embeddings have their representations pushed towards each other using a slightly different variation of the InfoNCE loss defined as

$$L_{\text{InfoNCE}} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=0}^{K} \exp(\text{sim}(z_i, z_k)/\tau)}. \tag{3.12}$$

where parameter τ controls the temperature of the softmax, the $\text{sim}(\cdot)$ function denotes a vector similarity function such as the cosine similarity

$$\text{sim}(u, v) = \frac{u^T v}{\|u\|\|v\|} = \frac{\sum_{i=1}^{n} u^T v}{\sqrt{\sum_{i=1}^{n} (u)^2}\sqrt{\sum_{i=1}^{n} (v)^2}}, \tag{3.13}$$

and K refers to the size of the queue containing negatives.

A queue containing 65 536 past representations is used to supply negative examples. Since MoCo does not create context vectors, a positive pair is composed of two augmented views of the same image (v_i, v_j).

The momentum encoder f_m does not receive gradient updates. Instead, it is updated as a momentum-based moving average of the query encoder training parameters. The update equation is defined as

$$W_m = \alpha W_m + (1 - \alpha) W_q, \tag{3.14}$$

where W_q and W_m refer to the training parameters of f_q and f_m respectively and $\alpha \in [0, 1)$ is the momentum coefficient, which controls by how much the f_m encoder takes new information from the f_q encoder at each time step.

The momentum encoder has the advantage of decoupling the negatives from the mini-batch. It allows scaling the framework without resorting to extremely large batch sizes, significantly reducing the memory footprint and processing time. MoCo representations were able to surpass supervised pre-trained representations in seven detection/segmentation tasks on PASCAL VOC [48] and MS COCO [77].

Chen et al. [57], followed by Chen et al. [78], further developed the ideas presented in MoCo, and introduced A Simple Framework for Contrastive Learning of Visual Representations (SimCLR). Combining simplicity and elegant design, SimCLR managed to reduce the gap between supervised and unsupervised pre-trained representations by a considerable margin. Driven by recent successes such as [79], SimCLR's contrastive learning framework highlights three main contributions: (1) large batch-sizes, (2) a composition

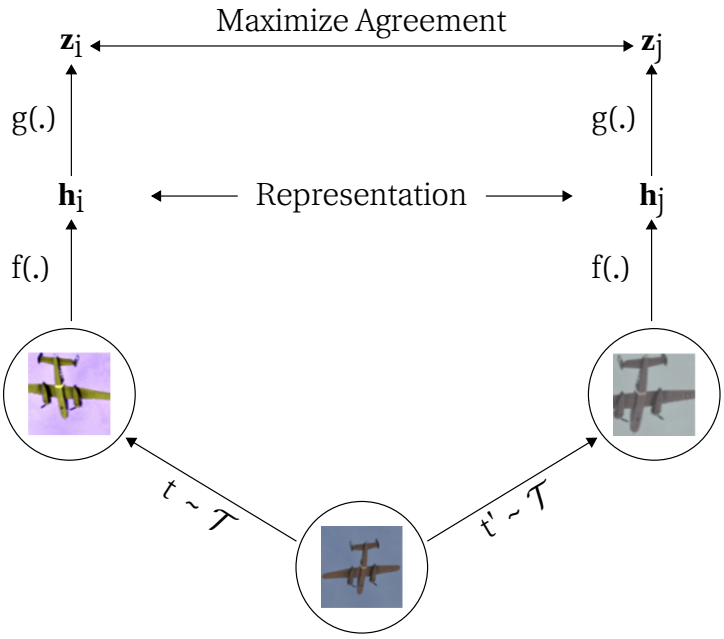

Figure 3.5. To create positive pairs, SimCLR randomly transforms an input image by sampling a random transformation τ from a set of transforms T. Each view is independetly processed by the same network and the system is trained to maximize agreement between the corresponding output representations.

of data augmentations, and (3) a non-linear projection head between the representation and the contrastive objective. SimCLR learns representations by maximizing agreement between augmented views of the same image. Different from MoCo, SimCLR uses two siamese ResNet-50 encoders to produce low-level representations from the two views. The embeddings are passed through a non-linear Multilayer Perceptron (MLP) that produces the pair of (anchor and positive) embeddings for the contrastive objective. Figure 3.5 depicts a visual description of the SimCLR learning framework.

Unlike previous work such as MoCo [56] and PIRL [54], which use an additional momentum encoder and a queue structure to process and create negative samples, SimCLR extracts negatives from within the training batches. Hence, to mine a sufficient number of negatives, SimCLR uses batch-sizes as large as 8192 records, which provides a total of 16 382 negative examples per positive pair. In general, given a positive pair, the other $2(N-1)$ views within the batch are treated as negatives. They also showed that normalized embeddings are beneficial to contrastive learning, and carefully adjusting the temperature parameter τ in the InfoNCE loss improves the learned representations. SimCLR pre-trained representations broke the state-of-the-art benchmarks in linear classification and semi-supervised learning using 1% and 10% of labels.

Similar techniques used to learning representations on SimCLR were used to improve knowledge distillation models Chen et al. [78].

After SimCLR, many works have been proposed to improve upon its baseline. Tian et al. [80] conducted a comprehensive study to investigate the necessary properties that good views for contrastive learning should have. The proposed InfoMin principle describes two rules for constructing optimal views for contrastive learning. First, given a downstream task T, optimal views v_i and v_j should retain all the essential information from the original input image x_i that can solve task T. In other words, good views should

retain all the relevant task information from the original input. Second, good views should share as little MI as possible. If the views share too much MI, the performance of the learned representations on downstream tasks degrades. In this sense, the InfoMin principle defines an optimal middle ground where the MI among views is minimal, as long as it preserves the relevant information for a particular downstream task.

In this way, a good view is task-dependent, and the choice of downstream task heavily influences how to create optimal views. For instance, if the downstream task is classification, the positive pairs' representations should only contain information to discriminate between the classes. Therefore, data augmentations that corrupt the object class to a point where it is unrecognized are not desired. Also, if the representations encode information from irrelevant factors of variation, such as the background, these may hurt the performance on downstream tasks.

The authors proposed learning an invertible view generator to create optimal views following the InfoMin principle. The generator is trained to minimize the MI between views while maintaining the class-relevant information. After training, we can use the view generator to train a contrastive learning encoder from scratch. The work also proposes a composition of manually designed data augmentations that follow the InfoMin principle. Using these manually crafted transformations, a pre-trained ResNet-50 encoder beat previous baselines, such as SimCLR, achieving a new SOTA performance on the ImageNet linear evaluation protocol. Also, transfer learning to PASCAL VOC object detection and COCO instance segmentation outperforms supervised pre-trained representations.

Tian et al. [81] proposed a method based on contrastive learning training to approach the problems of (1) Neural Knowledge distillation, (2) Cross-model transfer, and (3) Ensemble distillation. For the first, the goal is to distill a larger network, called the teacher, into a smaller one, the student, while preserving the representation's quality as much as possible. In Cross-modal transfer, a larger network (the teacher) is trained in a supervised manner using a large corpus of annotated data. The goal is to transfer knowledge to another network (the student) trained in a different domain, possibly in a low data regime. Lastly, in Ensemble distillation, the purpose is to transfer the knowledge of an ensemble of N neural networks into a single student model. Here, during distillation, the contrastive objective encourages both models, the teacher, and the student, to map the same input to similar representations and different inputs to representations that lie far apart in the embeddings space. In other words, the contrastive loss maximizes a lower bound on mutual information between the teacher and student network. The method is trained with the InfoNCE loss, Equation (3.12), and the results on distillation managed to surpass the previous method proposed by Hinton et al. [82] that minimizes the Kullback–Leibler (KL) divergence between the teacher and student outputs.

Apart from minor differences, recently proposed architectures using the InfoNCE loss, Equation (3.12), are very similar. Nevertheless, contrastive learning methods significantly reduced the performance gap between supervised and unsupervised pre-trained representations for downstream task learning.

3.5 Non-Contrastive Self-Supervised Learning

Contrastive learning methods are one of the most effective techniques to learn representations in a label-free way. The contrastive loss is optimized such that the distance between positive pairs is minimized while the distance among negative pairs is maximized. On the contrary, non-contrastive SSL methods do not require negative pairs to learn representations. They solely approximate the embeddings of positive pairs in the feature space. Non-contrastive methods have recently achieved significant results for self-supervised representation learning. However, it is still not well understood why the joint embedding training of such loss functions does not collapse to a trivial solution where the embeddings are mapped to the same constant output. Recently, various works tried to understand how these methods can be stably trained without negatives [83, 84].

Grill et al. [85] proposed Bootstrap Your Own Latent (BYOL), a SSL model with a non-contrastive cost function for learning representations in the vision domain. The method learns representations by approximating different views of the same image in the feature space. However, unlike contrastive methods, it does not require negative samples in its loss function.

BYOL comprises an asymmetric joint-embedding architecture with two independent neural networks denoted as online and target networks. The online network is composed of three components, (1) a ResNet-50 encoder f_θ, a projection networks g_θ, and a prediction network q_θ. On the other hand, the target network only contains the encoder f_ξ and a projection network g_ξ. The encoder and projection head of the online and target networks have equal architecture and a different set of trainable weights.

A pair of views v_i and v_j, extracted from a source image x_i, is independently passed through each network. For each view, BYOL produces two low dimentional vectors $z_i = g_\theta(f_\theta(v_i))$ and $z_j = g_\xi(f_\xi(v_j))$ from the online and target networks respectively. Instead of comparing the two embeddings directly, BYOL takes a step further and uses an additional network q_θ to predict the output of the target network. Refer to Figure 5.5 b) for a visual description of the architecture.

Similar to He et al. [56], the weights of the target network ξ are updated as a moving average of the online network's weights θ.

BYOL optimizes the online network by minimizing the normalized mean square difference between the online and target output representations, as

$$L_{\text{MSE}} = \|\overline{q_\theta}(z_i) - \overline{z_j}\|_2^2, \tag{3.15}$$

where, similar to contrastive methods, the representations z_i and z_j are projected to a unit hypersphere (l2-normalized) before minimization. Different from the contrastive InfoNCE loss, Equation (3.12), the mean squared error loss only approximates representations of positive views.

Representations from a pre-trained BYOL encoder achieved SOTA performance in the ImageNet linear evaluation protocol. For semi-supervised evaluation, BYOL also achieves SOTA performance using a limited number of annotated observations.

Empirical tests demonstrated that BYOL is more robust to some of the main limita-

tions of SimCLR. BYOL works with smaller batch sizes without a significant decrease in performance, and it is more robust to the set of data transformations used to create the views.

An advantage of not having negatives is to avoid false negatives that may occur during contrastive optimization. Most of the solutions that optimize the InfoNCE loss pick negative samples using a random uniform distribution. Depending on the number of classes of the dataset, the probability of picking an image of the same class as the positive pair, and treating it as a negative, can be high. The normalized MSE loss function in Equation (3.15) only approximates feature vectors from views created from the same source image, which avoids the noisy signals from false negatives that might jumble the learning signal.

Note that most contrastive and non-contrastive SSL methods often use additional components to improve performance on downstream tasks. Popular extra components include a momentum-encoder, a queue, or a memory bank to simulate large batch sizes or provide a reasonable number of negatives. Simple Siamese Networks (SimSiam), proposed by Chen and He [84], presented a study describing the importance of the main learning components of previous SSL methods. Specifically, the investigation shows that a simple siamese architecture without such additional components and large batch sizes can learn representations as well as the current SOTA methods.

The work explores how to avoid trivial solutions when training join-embedding architectures in a self-supervised context and provides insights regarding why non-contrastive methods such as BYOL do not collapse during training. The study shows that additional momentum encoders He et al. [56] and Grill et al. [85] are not directly responsible for avoiding trivial solutions. Moreover, in the asymmetric architecture proposed by Grill et al. [85], the predictor network q_θ is a vital component for stabilizing training of non-contrastive methods. Without it, non-contrastive training collapses. Lastly, stopping the gradient flow from one of the branches of the join-embedding architecture is fundamental to avoid trivialities. Without the "stop-grad" operation, non-contrastive methods crumble to trivial solutions.

Based on these findings, the proposed SimSiam model learns representations by optimizing a loss function that minimizes the negative cosine similarity between the two embedding vectors, as

$$L_{\text{SimSiam}}(x_i, x_j) = -\frac{q_\theta(z_i)}{\|q_\theta(z_i))\|_2} \cdot \frac{z_j}{\|z_j)\|_2}. \tag{3.16}$$

Note that, if the representation vectors are normalized, which is the case for SimSiam, the loss in Equation (3.16) is equivalent to the l2-normalized MSE loss from BYOL, Equation (3.15). Similar to the BYOL architecture, SimSiam employs a non-simetric architecture with an encoder f_θ, a projection network g_θ, and a preditor network q_θ. However, these components share weights between the two views, i.e., it uses a pure siamese architecture instead of non-differentiable momentum networks, refer to Figure 5.5(c), for a graphical description.

Following previous algorithms based on deep clustering [16, 17, 86], Caron et al. [87] presents Swapping Assignments between Views (SwAV), a pretext task that merges ideas

from deep clustering and acSSL. The idea is to predict the probability distribution assigned to view v_i based on the probability distribution from view v_j. Since both views come from the same source image x_i, we want the two views to have similar probability distributions over the prototypes.

Different from other clustering methods for visual representation learning [19], SwAV learns clusters online. To avoid trivial solutions, the clustering assignment task is done under a balanced partition constraint, solved by the Sinkhorn-Knopp [32] transform algorithm. The Sinkhorn-Knopp algorithm avoids the trivial solution where the loss function is minimized by assigning all views to the same cluster. It acts as an optimal load balancer by uniformly assigning views across all available centroids.

SwAV employs a pure siamese architecture containing a ResNet-50 encoder f_θ, a projection head g_θ, and an additional assigner network (a simple linear layer) denoted here as h_θ. Refer to Figure 5.5(d) to a graphical overview. The assigner function h_θ learns a set of K prototypes $C = \{c_1, c_2, ..., c_K\}$.

Similar to other SSL methods, the two views v_i and v_j are mapped to a low-level representation $z_i = g_\theta(f_\theta(v_i))$ and $z_j = g_\theta(f_\theta(v_j))$ and subsequently projected to a unit hyper-sphere via l2-normalization of the embddings. The assigner network receives the views's representations as input and produces respective codes $q_i = h_\theta(z_i)$ and $q_j = h_\theta(z_j)$. The codes can be interpreted as non-normalized probabilities indicating the likelihood of a given view to belong to each of the K prototypes.

For a pair of codes q_i and q_j, one represents the prediction, and the other represents the target. First, the code q_j is passed through the Sinkhorn-Knopp algorithm to be used as targets, while the prediction code q_i is normalized using a temperature softmax. Then, the cross-entropy loss

$$\ell(z_i, q_j) = -\sum_k q_j^{(k)} \log(p_i^{(k)}), \quad \text{where} \quad p_i^{(k)} = \frac{\exp(\frac{z_i^T c_k}{\tau})}{\sum_{k'} \exp(\frac{z_i^T c_{k'}}{\tau})}, \tag{3.17}$$

is optimized to tune the parameters of the system.

The cross-entropy operates on the Sinkhorn optimized codes (from one view) and the probabilities output by the assigner (from another view). This operation is symmetrized across the representations to form the swapped prediction task, defined as

$$L_{\text{SwAV}} = \ell(z_i, p_j) + \ell(z_j, p_i). \tag{3.18}$$

SwAV do not use additional momentum encoders and huge memory queues. It still simulates larger batch sizes by storing previously computed batch representations.

The ideas of SwAV were put to the test in a challenger context. SElf-supERvised (SEER) Goyal et al. [88] is a scaled version of SwAV designed to learn visual representations in the wild. Instead of pretraining on the carefully curated ImageNet-1k dataset, the SEER SSL model trains a very large RegNetY [89] model containing 1.3 billion parameters on 1 billion randomly chosen images from social media platforms. SEER demonstrated that SSL models can learn generalizable representations from unbounded and completely non-curated datasets. Moreover, following the recent trends of training very large Large

Language Models (LLMs), large self-supervised models also demonstrate excellent performance as one-shot learners for various downstream tasks, confirming similar results presented by Chen et al. [78].

Following a different approach, Zbontar et al. [90] proposed Barlow Twins, a non-contrastive SSL method that learns unsupervised representations without the "non-stop" gradient operation from previous implementations. Barlow Twins trains a join-embedding architecture containing an encoder f_θ and a projection head g_θ. Unlike prior methods, Barlow Twins avoids collapsed solutions by maximizing the information within the embeddings, which is done by forcing the empirical cross-correlation matrix $\mathbf{C}_{ij}$, between the embeddings of the two views to be as close to the identity matrix as possible.

Based on the principle of "redundancy-reduction", Barlow Twins proposes a novel loss function

$$L_{\text{BarlowTwins}} = \sum_i (1 - \mathbf{C}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} \mathbf{C}_{ij}^2, \tag{3.19}$$

that combines two terms: (1) an invariance and (2) a redundancy reduction terms, and the cross-correlation matrix $\mathbf{C}$ is defined as

$$\mathbf{C}_{ij} \overset{\text{def}}{=} \frac{\sum_b z_{b,i}^a z_{b_j}^p}{\sqrt{\sum_b \left(z_{b,i}^a\right)^2} \sqrt{\sum_b \left(z_{b,j}^p\right)^2}} \tag{3.20}$$

Note that Barlow Twins' loss function operates in the components of the embedding vectors. The first term of the loss in Equation (3.19) forces the diagonal values of the cross-correlation matrix $\mathbf{C}$ to have a strong correlation value of 1. On the other hand, the second term of Equation (3.19), forces the off-diagonal values of $\mathbf{C}$ to be non-correlated, i.e., a correlation value of 0. In other words, the redundancy reduction term forces the components of the embeddings to be non-correlated.

3.6 Broader Impact

Table 3.1 highlights the performance of several contrastive and non-contrastive SSL methods pre-trained on the ImageNet-1k dataset. Besides the steady increase in performance, Table 3.1 reveals how small the gap between supervised and unsupervised pre-trained representations have become.

Moreover, Table 3.1 demonstrates some of the limitations as well as architectural choices of current SSL methods for vision. One of such limitations is the reliance that current SOTA algorithms have on enormous batch sizes. The best methods require batch sizes as large as 4096 observations.

To train SSL methods with such memory budgets, one needs to have access to very expensive computational workstations with numbers of Graphics Processing Unit (GPUs) or Tensor Processing Units (TPUs). Table 3.1 also presents the performance of current SSL methods on semi-supervised benchmarks. Self-supervised pre-trained models are significantly more effective in learning new tasks with very few annotated observations.

The techniques discussed in this chapter for learning representations using contrastive and non-contrastive SSL models can be applied to different modalities. Currently, con-

Table 3.1. Performance of various contrastive and non-contrastive SSL methods for vision. The models were pre-trained on the ImageNet-1K dataset without labels. The encoder is a ResNet-50 (RN50) without the fully-connected layers. The performance of the learned representations is measured in terms of linear separability using the linear evaluation protocol, Subsection 4.2.1, and semi-supervised learning, Subsection 4.2.2.

				Linear eval.		Label frac.			
Method	**Enc.**	**Epochs**	**Batch size**	**Top-1**	**Top-5**	**1%**	**10%**	**Mem. bank**	**Mom. enc**
Supervised [91]	RN50	100	256	76.5		48.4	80.4		
MoCo-v2 [75]	RN50	200	256	67.5				65536	✓
SimCLR [57]	RN50	200	8192	66.6					
SwAV-MC [16]	RN50	200	256	72.7				3840	
InfoMin [80]	RN50	200		70.1	89.4				
SimSiam [84]	RN50	200	256	68.1					
OBoW-MC [92]	RN50	200	256	**73.8**		82.9	90.7	8192	✓
results of longer unsupervised training									
PIRL [54]	RN50	800	1024	63.6		57.2	83.8	32000	
MoCo-v2 [75]	RN50	800	256	71.1					✓
SimSiam [57]	RN50	1000	256	71.3					
SeLa-v2	RN50	400	4096	71.8					
DeepCluster-v2	RN50	400	4096	74.3					
SimCLR [57]	RN50	1000	4096	69.3	89.0	75.5	87.8		
SwAV-MC [16]	RN50	800	4096	**75.3**		78.5	89.9	3840	
BYOL [85]	RN50	1000	4096	74.3	91.6	78.4	89.0		✓
InfoMin [80]	RN50	800		73.0	91.1				
Barlow Twins [90]	RN50	1000	1024	73.2	91.0	79.2	89.3		

trastive learning models are SOTA for learning speech representations from unlabeled audio signals Baevski et al. [53]. After self-supervised pre-training, it is possible to learn many downstream speech recognition tasks without extensive annotated audio data.

Another very promising modality for the application of SSL is medical imagery. Creating a large supervised dataset can be very expensive, even for the general case. For the healthcare domain, however, this cost is even more pronounced. Pre-training self-supervised encoders on various medical data modalities increase the performance on downstream tasks and reduce the reliance on professionals to annotate large quantities of data [93].

Moreover, since self-supervised pretext tasks are independent of the learning architecture, recent methods using Transformers [5, 94], have demonstrated a very prospective path to learn richer unsupervised representations from visual data [95].

Chapter 4

Datasets and Evaluation Protocols for Self-Supervised Learning

Having covered a substantial history of recent Self-Supervised Learning (SSL) methods for Computer Vision (CV) and established the key ideas behind the success of such unsupervised algorithms, in this chapter, we introduce part of our experimentation setup. Here, we highlight the datasets used to pre-train the SSL algorithms presented in this thesis. We also describe the protocols to measure and compare the quality of the representations learned by self-supervised methods.

Up to this point, we have covered many approaches to self-supervised training, from algorithms that optimize self-prediction tasks such as the rotation prediction task, Section 2.1.4, to recent techniques that optimize contrastive and non-contrastive cost functions. To recall, the goal of SSL is to pre-train an encoder network using non-labeled data, hoping that the representations learned by the encoder will be generalizable across different downstream tasks. When presenting many of the self-supervised algorithms, in Chapters 2 and 3, for many of them, we stated that they achieved a certain level of performance in downstream tasks. This chapter ties the concepts of performance measures of self-supervised models. We describe how current methods, including our own, are assessed and explain which datasets we used to perform such evaluations.

4.1 Datasets

Many standard CV datasets are used to train representation learning models, regardless of being supervised datasets or not. For self-supervised representation learning, classic datasets such as the ImageNet are used to pre-train self-supervised algorithms. Note that pre-training is done in a label-free way, i.e., we usually pretend we do not have access to human labels.

Training on a supervised dataset such as the ImagetNet is useful for evaluation purposes. Once a self-supervised encoder is pre-trained, we can use the validation/test sets to train a downstream classification task that will serve as an assessment for the quality of the learned representations.

It is important to note that even though we pretend not having access to the human

labels associated with the training images, it does not make the dataset completely free of human biases. Datasets such as the ImageNet encode human biases in different ways. One such source of human biases in the ImageNet dataset regards the position of the main objects on the images. Naturally, when humans take pictures of a particular object, we tend to position the object in question to the center of the picture. Supervised and unsupervised methods can explore this hidden bias to figure out the position of the object or its class label. Indeed, Mo et al.'s [96] and Purushwalkam and Gupta's [97] works have demonstrated that current self-supervised methods based on instance discrimination explore hidden human biases in the data to improve performance on downstream tasks.

4.1.1 Pre-training and Validation

Below, we enumerate the classic CV datasets we used for pre-training and evaluating SSL algorithms.

CIFAR-10/100: For small to medium classes, we used the CIFAR-10/100 dataset family. Each dataset contains 60 000, 32×32 RGB images, with 50 000 images for training and 10 000 for testing. CIFAR-10 has 10 classes with 6000 images per class, and CIFAR-100 has 600 images for each of the 100 classes.

STL-10: We used the STL-10 unsupervised and supervised sets for fine-tuning and semi-supervised experiments. The unsupervised portion contains 100 000, 96×96 RGB images, and the supervised set has 5000 images from 10 classes.

Downsampled ImageNet: For datasets with a significant number of classes, we used the Tiny-ImageNet and a downsampled version of the original ImageNet dataset, referred to as DS ImageNet. Tiny-ImageNet contains 100 000 64×64 RGB images balanced across 200 different categories. The downsampled version of ImageNet has 1000 classes with 1 281 167, 64×64 RGB images.

ImageNet-1k: Since the majority of SSL methods report results using the ImageNet-1k dataset, we also include it in our validation protocols. The dataset contains 1000 object classes and includes 1 281 167 training RGB images of various sizes and 50 000 validation images.

4.2 Feature Evaluation for Self-Supervised Learning

The main objective of representation learning is to learn compressed and transferable representations across different tasks and domains. In this context, protocols to evaluate the quality of self-supervised representations focus on two main objectives: linear separability and transfer learning to downstream tasks. Here, a downstream task describes the task that we care about the most. In Natural Language Processing (NLP), for instance, state-of-the-art (SOTA) methods may learn word embeddings by solving a pretext task such as predicting the following words in a sentence. However, once the embeddings are optimized, we are interested in using these features to solve relevant tasks with real-world applications such as sentiment analysis, summarization, named entity recognition, and others.

Similarly, self-supervised methods learn representations by optimizing pretext tasks such as rotation prediction, jigsaw puzzle, or instance discrimination. However, the tasks we are most interested in, i.e., the downstream tasks, are the ones that solve real-world problems such as object detection, segmentation, or depth estimation. For this reason, the standard protocol to assess the quality of self-supervised features is by measuring how good they are when solving downstream tasks.

There are two main categories for evaluating self-supervised features. In the first, we fine-tune the pre-trained self-supervised model to solve a specific downstream task. In the second, we use a pre-trained encoder as a fixed feature extractor to provide salient features that can transfer prior knowledge to solve related problems.

In both scenarios, the evaluation process uses labeled data and supervised tasks such as classification, detection, and segmentation as proxies to measure the quality of the unsupervised representations. In other words, metrics such as classification accuracy, Intersection over Union (IoU), or mean Average Precision (mAP) are used as proxies to measure the quality of the representations used to learn respective downstream tasks.

4.2.1 Linear Evaluation Protocol

One of the most popular ways of measuring the quality of self-supervised representations is by assessing the linear separability property of the embeddings. This protocol, commonly known as linear evaluation, can be done using different methods.

A popular way of measuring linear separability is by training a linear classifier on top of the frozen representations extracted from the self-supervised encoder. The linear classifier can be a Support Vector Machines (SVMs) with a linear kernel or a Linear Regression model. The linear models are optimized using features from the training portion of the dataset, and classification accuracy on the held-out test set is reported as a measure of quality for the representations.

An alternative and computationally cheaper strategy to measure linear separability is training a K-Nearest Neighbors (KNN). The model is trained on top of the frozen representations from the training set and evaluated using the test set.

The linear evaluation protocol assumes that good representations should transfer well to other tasks with minimal effort, without non-linear complex models. Moreover, it considers the manifold hypothesis, which states that natural high-dimensional data in the real world lie in low-dimensional manifolds within the original high-dimensional space. In other words, the protocol tries to evaluate the linear subspaces in which high-dimensional non-linear data can be expressed.

4.2.2 Semi-Supervised Learning

Another commonly used protocol to evaluate self-supervised representations is to use semi-supervised benchmarks. In this scenario, the pre-trained encoder is either used as a frozen feature extractor or fine-tuned using small quantities of labeled data. Classic setups usually take 1% and 10% of labeled examples from the ImageNet training data in a class-balanced manner. The split produces two different training datasets with approximately

12.8 and 128 images per class. In most situations, we append a task-specific model on top of the self-supervised pre-trained encoder and use a small subset of labeled data to optimize the system. In the case of fine-tuning, both components are jointly updated. Otherwise, only the task-specific model receives gradient updates, and the encoder remains fixed. This evaluation protocol emphasizes that a good set of features should not require large quantities of human-annotated data to learn downstream tasks.

4.2.3 Cross-Domain Transfer

Lastly, self-supervised representations can also be evaluated based on how good they are at solving tasks from different domains. The idea is to measure if the features learned on one specific dataset are generalizable across different tasks or domains instead of being dataset-specific. The motivation lies in the belief that good representations should transfer well to several different domains. Usually, we use a self-supervised ImageNet pre-trained encoder in a transfer learning setup, and possibly fine-tuned it, on different datasets such as Cifar10 [43], Food101 [98], Birdsnap [99], Cars [100], or VOC2017 [101]. One reports performance using the held-out test set with the respective standard metrics for each case as a proxy for quality.

Chapter 5

Consistent Assignment for Representation Learning

With the fundamental concepts and ideas of existing Self-Supervised Learning (SSL) methods properly introduced, in this chapter, we present Consistent Assignment for Representation Learning (CARL). This work has been presented by Silva and Ramírez Rivera [102] in the Energy-Based Models Workshop at ICLR2021 as an ongoing work and subsequently published at the 37th ACM/SIGAPP Symposium on Applied Computing (SAC'22). CARL is an unsupervised learning method for visual representation learning that combines ideas from self-supervised contrastive learning and deep clustering. By viewing contrastive learning from a clustering perspective, CARL learns unsupervised representations by learning a set of general prototypes that serve as energy anchors to enforce different views of a given image to be assigned to the same prototype. Unlike contemporary work on contrastive learning with deep clustering, CARL proposes to learn the set of general prototypes in an online fashion, using gradient descent without the necessity of using non-differentiable algorithms or K-Means to solve the cluster assignment problem. CARL surpasses its competitors in many representation learning benchmarks, including linear evaluation, semi-supervised learning, and transfer learning.

5.1 Introduction and Motivation

Unsupervised visual representation learning focuses on creating meaningful representations from data and inductive biases. As discussed in Chapter 3, methods based on siamese neural networks [103] and contrastive loss functions [56, 57] have significantly reduced the gap between supervised and unsupervised based representations. Indeed, for some downstream tasks, unsupervised-based representations already surpass their supervised counterparts [87]. In Computer Vision (CV), approaches to unsupervised representation learning can be categorized into three groups: (1) contrastive learning methods using instance discrimination, (2) clustering-based methods, and (3) a mixture of the two.

Recent unsupervised representation learning methods rely on contrastive learning, Section 3.4. These methods optimize an instance discrimination pretext task where each image and its transformations are treated as individual classes. They compare feature

vectors of individual images intending to organize the feature space such that similar concepts are placed closer while moving different ones farther.

On the other hand, traditional clustering methods aim to learn the data manifold by comparing groups of features that share semantic structure based on a distance metric. When combined with deep learning, clustering methods are often designed as two-step algorithms. First, the complete dataset is clustered, and then the meta clustering information, e.g., prototypes and pseudo-labels, are used as supervisory signals in a posterior optimization task [16–18, 86].

Recent work has attempted to combine the benefits of contrastive learning and clustering [19, 87]. In particular, Expectation-Maximization (EM) approaches alternate between finding the clusters and maximizing the Mutual Information (MI) between the embeddings and the cluster centroids [19]. Inspired by them, our work merges the benefits of both approaches by bridging the gap between clustering and contrastive learning. On one hand, we use unsupervised clustering dynamics to generate robust prototypes that organize the feature space. On the other, we use contrastive learning to compare the distributions of the views' assignments w.r.t. the clusters. Our experiments show that we can learn unsupervised visual representations that outperform existing methods by mixing both approaches.

We can think of contrastive learning as learning clustered representations at the image level. However, given the nature of the task, these clusters fail to capture semantic information from the heterogeneous unknown classes since the learned clusters only comprise representations from synthetic views of an image. Moreover, since contrastive learning methods handle different images as negatives in the training process, even if two given images share the same class information, their representations will be pushed farther apart from each other. In the end, each image will have its own cluster structure.

5.2 Proposal

We propose an alternative method to learn high-level features by clustering views based on consistent assignments. Unlike previous work that uses K-Nearest Neighbors [104] or K-Means [105] as priors to enforce (learn) a cluster mapping, our method learns the prototypes online. While contrastive learning with instance discrimination [56, 57] poses a classification pretext task at the image embedding level, we propose a pretext task that operates on the assignment of views to a set of clusters. Rather than directly maximizing similarities between image embeddings, we force the distribution of positive views' assignments to be consistent among a set of finite learnable prototypes. If the number of prototypes equals the number of observations in the dataset, we would be forcing each cluster only to contain synthetic views of a given observation. This is equivalent to contrastive learning with instance discrimination. However, if we set the number of prototypes to be smaller than the number of observations in the dataset, by the pigeonhole principle, the learned prototypes will not only cluster different views of an image together, but they will also contain representations of different images that are similar enough to be assigned to the same cluster.

Our proposal, Consistent Assignment for Representation Learning CARL, re-frames Contrastive Learning through a clustering perspective to learn robust representations (Section 5.4). CARL builds distributions of the similarities between the prototypes and the image's views (Section 5.5). To avoid collapsing the representation in a subset of clusters, we impose an uninformative prior to CARL's prototypes (Section 5.5.1). CARL jointly optimizes similarity to the learned prototypes and the uninformative prior (with a decay schedule for learning). Figure 5.1 illustrates the overall idea.

5.3 Contributions

Regarding our contributions, (i) we propose a learning framework that leverages current contrastive learning methods with clustering-based algorithms to improve the learned representations. Unlike contemporary work, our method proposes to learn the clusters' assignments in an online fashion using gradient descent with no need for pre-clustering steps, e.g., K-Means or non-differentiable methods to solve the clustering assignment problem, such as the Sinkhorn-Knopp algorithm [32]. (ii) We contrast high-level structures (the distributions of the views over the cluster assignments) instead of low-level ones (such as the representations). Reasoning in this high-level space gives the representations more robustness that translates to better performance in downstream tasks. And, (iii) unlike contrastive learning methods that can be viewed as learning clusters containing only synthetic transformations of a given image, our learned prototypes do not need to hold the semantics of the data but rather become energy anchors that self-organize the space to learn better representations. Moreover, the proposed loss function does not require a more extensive set of negative representations, which avoids the common problem of treating representations of the same class as negatives.

5.4 Contrastive Learning from a Clustering Perspective

Let $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ be a dataset containing N unlabeled images. And, let a view $v_i = T(x_i)$ of an observation x_i be the application of a stochastic function T that is designed to change the content of x_i subjected to preserving the task-relevant information encoded in it. In practice, we can create as many views as needed by applying the stochastic function T. Recent contrastive learning methods learn visual embeddings by solving an instance discrimination pretext task that is usually optimized using the InfoNCE loss [71] defined as

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp\left(\text{sim}\left(z_i^a, z_i^+\right)/\tau\right)}{\sum_j^M \exp\left(\text{sim}\left(z_i^a, z_j\right)/\tau\right)}, \tag{5.1}$$

where z_i^a and z_i^+ are anchor and positive representations taken from an encoder function $f(\cdot)$ such that $z_i^{(\cdot)} = f\left(v_i^{(\cdot)}\right)$, τ is the temperature parameter, and $\text{sim}(\cdot,\cdot)$ is a similarity function, e.g., the cosine similarity. From a clustering perspective, the InfoNCE loss, Equation (5.1), is minimized when all possible variants $\left\{v_i^j\right\}_j$ of an image x_i are clustered

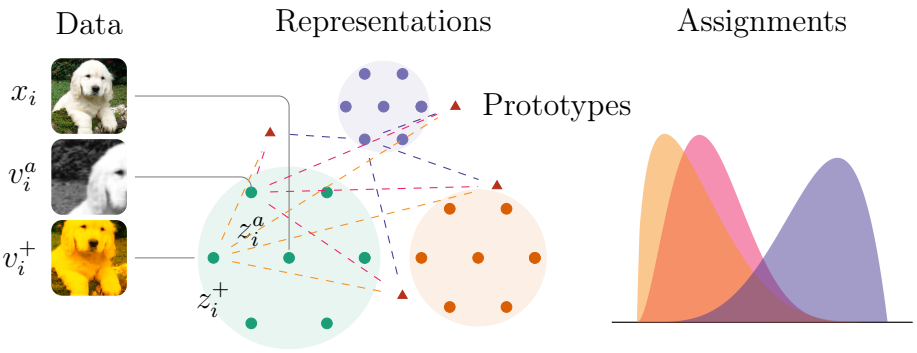

Figure 5.1. The data is first augmented and then encoded into their representations. We learn a set of prototypes that serve as anchors to compute a distribution of assignments. For a positive (augmented) pair of points and for the rest of negatives we compute their agreement to the prototypes. We collect these comparisons as distributions that we later compare. We expect that the positive examples will have similar distributions over the assignments (cf. orange and pink distributions), while the negatives will not (cf. purple distribution).

into the same prototype while representations from within a cluster are far apart from representations of other clusters.

We propose an approach where, instead of comparing against other instances [56] or prototypes of the classes [19], we learn a set of K general prototypes $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$, $K \ll N$, against which we compare the views to determine their similarity. Instead of maximizing a similarity function between positive embeddings of different views, as contrastive learning methods do, we maximize the similarity between assignment vectors of positive views to our general prototypes to promote consistency and confidence when assigning views to clusters. That is, views must agree with high confidence in their cluster assignments. Our method learns the prototype matrix $\mathcal{C}$ online using gradient descent, and it does not require any pre-clusterization step as is typically the case for clustering-based representation learning algorithms [16, 19]. Moreover, since the training process is unsupervised, to avoid trivial solutions, where all images are assigned to only a handful of prototypes, we enforce a non-informative prior over the class distribution of the views to guide the learning process.

5.5 Learning Representations by View Assignment

As highlighted as one of the fundamental ideas behind current self-supervised methods, we treat augmented versions of a given image as views and use them as positive examples for optimization, refer to Section 3.2. Our objective is to transform two positive samples into a distribution of their likelihood to belong to a set of K clusters. To do so, we encode each view through an encoder function $z_i = f(v_i) \in \mathbb{R}^d$. The encoder $f(\cdot)$ comprises a backbone convolutional neural network, such as a ResNet [76], followed by a non-linear Multilayer Perceptron (MLP) head.

Our objective is not to cluster the data in an unsupervised manner but rather to learn a set of prototypes that will serve as anchors to differentiate views. We hypothesize that similar views should have similar assignments w.r.t. the prototypes. Hence, to convert the representations into these assignments, we first compare the representation z_i against

all the prototypes to obtain an energy distribution

$$q_i[j] = \langle z_i, c_j \rangle, \tag{5.2}$$

where $q_i[j]$ is the j-th element of the energy distribution q_i for the i-th view. We learn the set of prototypes through a linear layer. To get a distribution of a given view over all prototypes, we normalize the energy using the softmax function and obtain the posterior probability distribution, i.e., the probability of assigning the view v_i to the cluster k. Hence, our normalized probability for the k-th class given our view v_i is

$$p_i[k] = P(i \text{ assigned to } k \mid v_i) = \frac{\exp(q_i[k])}{\sum_{j=1}^{K} \exp(q_i[j])}, \tag{5.3}$$

where $q_i[j]$ denotes the j-th element of the i-th un-normalized vector output of the classifier for the respective view.

As mentioned before, our objective is to contrast the distributions of two views' likelihood w.r.t. the clusters. In CARL, the encoder and assigner operate in a siamese setup where a pair of views from a given sample is independently transformed in its corresponding distribution. To ensure the similarity between the views, we optimize the views' distributions p_i^a and p_i^+ over the clusters in $\mathcal{C}$, so that the two distributions are consistent with one another. In other words, by learning a consistent assignment of views over the clusters, a given prototype will be invariant to augmented versions of an input sample. Moreover, because the number of prototypes is smaller than the number of observations in the dataset, the clusters will contain different observations that share similarities in the embeddings space.

We compute the similarity between the views' distributions as their dot product

$$\mathcal{L}_c = -\frac{1}{B} \sum_{i}^{B} \log \langle p_i^a, p_i^+ \rangle, \tag{5.4}$$

where B is the size of a minibatch over which we are aggregating the samples. In the ideal case of two one-hot vectors signaling the same perfect assignment, the dot product above yields its maximum value of one, and the negative log is minimized.

5.5.1 Preventing Trivial Solutions

Only forcing different views, v_i^a and v_i^+, to have the same cluster assignment using our consistency loss, Equation (5.4), leads to finding a trivial solution where all representations z_i end up assigned to the same cluster (or just to a handful of them). To prevent such triviality, we force the distribution over the classes, $\mathcal{P}$, to be uninformative by minimizing the Kullback-Leibler divergence (KL-divergence) w.r.t. a uniform distribution, U. Our regularization is

$$\mathcal{L}_{\text{KL}} = \text{KL}(\mathcal{P} \parallel U) = \log(K) + \sum_{c \in C} \hat{p}^c \log\left(\hat{p}^c\right), \tag{5.5}$$

where $\hat{p}^c$ is the expected distribution over a minibatch of size B,

$$\hat{p}^c = \frac{1}{B}\sum_{i}^{B} p_i^c, \tag{5.6}$$

and p_i^c is the probability of a view v_i to be assigned to a cluster c where $c \in C$. In other words, we maximize the Shannon entropy of the average distribution of the predictions. We can interpret the KL-divergence, Equation (5.5), as regularizing the encoder $f(\cdot)$ to encourage the approximate posterior, Equation (5.3), to be closer to the uniform distribution.

Minimizing the KL-divergence, Equation (5.5), will force the predictions for a given image view v_i to be spread across all clusters. Since we do not know the underlying class distribution in advance, the KL-divergence, Equation (5.5), acts as a prior where we assume that the observations $\mathcal{X}$ are uniformly assigned among all K prototypes.

By combining the consistency assignment loss, Equation (5.4), with the KL-divergence regularization, Equation (5.5), we obtain our final learning objective

$$\mathcal{L} = \mathcal{L}_c + \lambda_e \mathcal{L}_{\mathrm{KL}}, \tag{5.7}$$

where λ_e is an epoch-dependent function that returns a scalar that prevents mode collapse at the beginning of training. We observed that training is very susceptible to such collapsing to a single assignment if not regularized. However, in practice, we noticed that keeping a large fixed value of λ_e during training also prevents the encoder from learning more complex representations. Thus, we recommend a function λ_e that decreases as training progresses. In theory, any decay schedule, such as an exponential or cosine, could be used. We propose a linear decay schedule

$$\lambda_e(a, b) = \begin{cases} b - \frac{b-a}{E} e & \text{if } e \leq E, \\ a & \text{otherwise,} \end{cases} \tag{5.8}$$

where b and a denote the start and ending values of the decay, E represents the number of epochs in which the decay will happen, and e is the epoch counter.

We noticed that mode collapse can happen in two scenarios: (1) if the regularizer is not added to the final loss, which is equivalent to setting the KL weight penalty $\lambda_e(0, 0)$, or (2) if the KL weight penalty λ_e is set to a value below a certain threshold, cf. Figure 5.2. In both situations, CARL finds suboptimal solutions and learns bad representations. In practice, we found that slowly decreasing the KL weight penalty throughout training can work twofold since it increases the quality of the representations and prevents such trivialities during optimization. Refer to Section 5.6.1 and Figure 5.2 for an in-depth analysis.

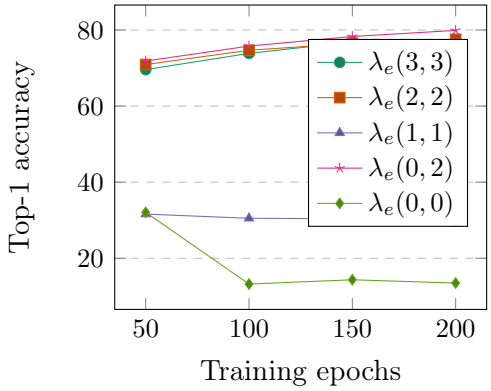

Figure 5.2. Effect of the uninformative prior's scheduling λ_e on the overall performance in STL-10. Notice that a linear decay scheduling outperforms its constant counterparts. Two situations may result in non-optimal solutions: (1) a lower value of λ_e, and (2) not having the KL regularizer in the final loss.

5.6 Hyperparameter Exploration

In this section, we evaluate the effects of the primary hyperparameters of our method. For exploring hyperparameters, we learn representations using a ResNet-18 backbone trained for 150 epochs, and the KL weight penalty λ_e is linearly decayed over the first $E = 100$ epochs. To evaluate the multiple experimental setups, we train linear classifiers on top of the encoder's frozen features following the linear evaluation protocol proposed by He et al. [56] and report average Top-1 accuracy over three independent runs.

5.6.1 Does Decreasing the KL Weight Penalty Improves Representation Learning?

The hyperparameter λ_e controls the contribution of the KL regularization, Equation (5.5), to the consistency loss, Equation (5.4). Especially at the beginning of training, a higher contribution for the KL term avoids mode collapsing, where the network optimizes the consistency loss, Equation (5.4) by assigning all observations to the same prototype. Van Gansbeke et al. [106] make similar claims for the entropy regularization in their SCAN-loss [106], and suggest a high (constant) value for the scalar hyperparameter λ_e to avoid such trivialities.

We hypothesize that keeping a high value of λ_e throughout training prevents the network from learning complex features. To verify this hypothesis, we trained CARL on the STL-10 [42] unsupervised dataset for 200 epochs. We measure the performance by training a linear classifier on top of the frozen features of the ResNet-18 backbone. We linearly decay the magnitude of the λ_e hyperparameter, following Equation (5.8), from $b = 2$ to $a = 1$ over the first $E = 100$ epochs instead of keeping it constant for one of the experiments. As shown in Figure 5.2, we observe that the quality of the representations learned by CARL benefits from decreasing the contribution of the KL regularization. However, if the λ_e is decreased below a certain threshold, mode collapse may happen.

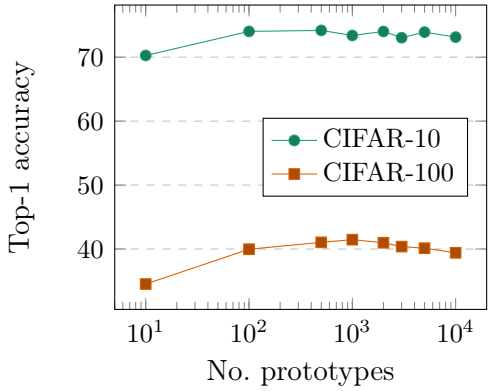

Figure 5.3. Effect of learning a different number of prototypes on the quality of the representations. Empirical tests suggest an inverse U-shape curve where the optimal number of prototypes lies near one order of magnitude w.r.t. the actual number of classes of the dataset.

5.6.2 Does the Number of General Prototypes Influence the Quality of the Representations?

To evaluate the effect of learning a different number of prototypes, we trained CARL with ResNet-18 backbones on the CIFAR-10/100 datasets [43] for 150 epochs. For all experiments, the KL weight term λ_e starts as $b = 4$ and is linearly decreased to $a = 1.5$ over the first $E = 100$ epochs. Figure 5.3 shows that over-clustering benefits the quality of the learned representations and that the optimal number of general prototypes depends on the number of actual classes of the dataset. The experiments suggest an inverse U-shape for both datasets, and the optimal number of general prototypes lies near an order of magnitude w.r.t. the actual number of classes.

5.6.3 Does the Batch Size Improves the Representations Learned by CARL?

We regularize CARL using a non-informative uniform prior, Equation (5.5), applied over the input batch. This uniform prior prevents the views' assignment from collapsing to a single prototype. We work over the expected distribution of assignments instead of each individual one. This expectation raises the question of how large a sample (batch size) should be to model different numbers of general prototypes.

Table 5.1 shows the effect of training CARL with different batch sizes and their relationship with the number of general prototypes. For this experiment, we trained CARL on CIFAR-100 for 150 epochs. Note that to compensate for the reduced number of iterations (as we increase the batch size), we scale the learning rate following the recipe proposed by Goyal et al. [107].

With smaller batches, we can see that the performance of the representations learned by CARL degrades as the number of prototypes increases. On the other hand, large batch sizes tend to retain high performance even when we increase the number of prototypes to very large numbers. Indeed, the gap in performance due to the batch size can reach nearly 6 points of average accuracy when optimizing a downstream task. We hypothesize that this is a direct effect of the sample size used to estimate the entropy in Equation (5.5). In

Table 5.1. The effect of training CARL (on CIFAR-100) with different batch sizes and general prototypes.

	Number of prototypes						
Batch	10	100	500	1000	3000	5000	10000
64	**35.34**	38.72	38.60	37.64	35.98	35.75	34.12
128	35.25	39.43	39.50	38.97	38.38	38.59	36.33
256	34.46	40.31	40.51	40.88	39.98	38.83	39.18
512	35.01	40.91	**42.559**	41.23	41.06	40.61	38.65
1024	34.67	**41.259**	41.44	**42.049**	**41.479**	**41.548**	**40.689**

other words, as the distribution increases (with a higher number of general prototypes), the KL regularizer requires a larger batch size to better estimate the entropy and force it to be high, Equation (5.5).

5.7 Unsupervised Feature Evaluation

We evaluate CARL's representations extracted from a ResNet-18 encoder and compare its performance with different state-of-the-art methods using the linear evaluation protocol [56], cross-domain transfer, semi-supervised learning, and fine-tuning.

5.7.1 Linear Evaluation

We trained self-supervised models for 200 epochs, closely following their original hyperparameters. Refer to Section 5.8 for more details. Table 5.2 compare our results with previous approaches on CIFAR-10/100 [43], STL-10 [42], and on a downsampled version (DS) of the ImageNet dataset [108]. We compare the representations learned by CARL against two methods based on instance discrimination with contrastive learning, i.e., A Simple Framework for Contrastive Learning of Visual Representations (SimCLR) and Momentum Contrast for Unsupervised Visual Representation Learning (MoCo) version 2, against two clustering-based methods, i.e., Prototypical Contrastive Learning (PCL) and Swapping Assignments between Views (SwAV), and for completeness, we also include Bootstrap Your Own Latent (BYOL), which does not rely on negatives during optimization.

On average, CARL outperforms all major competitors in most setups and only stays behind on early training cases (100 epochs) in the STL-10 and DS ImageNet datasets. Unlike SimCLR and MoCo v2, our method does not use negative examples, nor does it require a momentum-based target encoder to prevent collapsing [85].

We adapted PCL and SwAV to use a ResNet-18 encoder and kept official hyperparameters. Most of these methods were only tested on large-scale datasets, such as the ImageNet [4], usually with very deep convolutional encoders. Thus, it naturally raises the question of how these methods would perform on smaller datasets, using faster and not-so-deep encoders, without enormous batch sizes and trained on modest Graphics Processing Unit (GPUs). Our experiments highlight some of the limitations of training these

data-hungry methods on limited resources.

5.7.2 Cross-domain transfer

To evaluate how well the representations learned by CARL transfer between different distributions, the first two columns of Table 5.3 show experimental results on cross-domain transfer. We took a self-supervised encoder trained on the CIFAR-10 dataset and used it as a feature extractor to train a linear classifier (for 50 epochs) to solve a 100-way classification task using the CIFAR-100 dataset (and vice versa). Representations learned by CARL outperform (on average) the main competitors in both setups.

5.7.3 Semi-supervised learning

The third and fourth columns of Table 5.3 report results on semi-supervised learning. First, we trained self-supervised models (for 200 epochs) on the unsupervised portion of the STL-10 dataset. Afterward, we fine-tuned the encoders (for 50 epochs) using different portions (1 % and 10 %) of labeled data from the supervised set of STL-10. Here, representations learned by CARL outperform competitors on the 10 % setup and perform on pair with SimCLR on the more extreme case of only 1 % of labeled data.

5.7.4 Fine-tuning

Lastly, similar to the semi-supervised evaluation, the fifth column of Table 5.3 shows results on fine-tuning the self-supervised encoder (trained on the unsupervised STL-10 for 200 epochs) using the entire supervised STL-10 dataset (for 50 epochs). CARL's representations outperform competitors by at most 2.43 % average accuracy.

Table 5.2. Results are reported using mean accuracy (percentage) and standard deviation over three runs (except for DS ImageNet where we report a single run) on the linear evaluation protocol.

	CIFAR-10		CIFAR-100		STL-10		DS ImageNet	
Method	100	200	100	200	100	200	100	200
Supervised	87.76 ± 0.07	89.40 ± 0.22	59.23 ± 0.28	60.59 ± 0.09	73.11 ± 0.24	74.76 ± 0.25	35.06	43.50
BYOL	68.90 ± 0.24	76.46 ± 0.37	37.26 ± 0.47	45.68 ± 0.16	76.58 ± 0.78	79.64 ± 0.20	28.12	30.96
SimCLR	72.64 ± 0.34	75.41 ± 0.47	41.64 ± 0.14	44.94 ± 0.11	**77.36 ± 0.39**	80.57 ± 0.66	**28.20**	30.32
MoCo	65.57 ± 0.65	71.62 ± 1.04	39.12 ± 0.43	44.35 ± 0.41	71.50 ± 0.96	75.46 ± 1.28	24.82	28.67
PCL	66.37 ± 0.23	71.29 ± 0.54	35.97 ± 0.53	40.1 ± 07.6	69.61 ± 0.93	70.56 ± 0.41	22.60	26.18
SwAV	72.75 ± 0.65	75.93 ± 0.48	39.98 ± 0.83	43.20 ± 0.35	70.89 ± 0.55	74.35 ± 0.47	18.39	25.31
CARL	**73.39 ± 0.31**	**78.94 ± 0.52**	**42.91 ± 0.26**	**48.85 ± 0.79**	76.9 ± 01.2	**81.95 ± 0.10**	24.62	**31.86**

5.8 Implementation Details

For evaluating CARL's performance against its competitors, all methods were trained for 200 epochs using the same cosine learning rate scheduler and the same batch size of 256 observations. If not specified otherwise, we kept all the hyperparameters fixed as the original implementations.

Table 5.3. Results are reported using mean accuracy (percentage) and standard deviation over three runs. Best results in bold. Cross-Domain Transfer: We trained CARL on the unlabeled CIFAR-10 dataset and performed linear evaluation on the labeled CIFAR-100 (10 → 100), and vice versa (100 → 10). Fine-tune: We trained CARL on the unlabeled STL-10 dataset and fine-tune it on the labeled STL-10.

	Cross Domain Transfer		Semi-supervised		Fine-tuning
Method	C-10 → C-100	C-100 → C-10	1 %	10 %	100 %
Supervised	38.11 ± 0.41	68.54 ± 0.42	52.97 ± 0.07	69.87 ± 0.03	76.11 ± 0.01
BYOL	43.47 ± 0.20	67.23 ± 0.45	36.64 ± 0.43	65.7 ± 09.8	85.72 ± 0.10
SimCLR	40.39 ± 0.22	67.4 ± 03.0	$\mathbf{53.66 \pm 0.11}$	73.92 ± 0.26	86.97 ± 0.03
CARL	$\mathbf{43.47 \pm 0.02}$	$\mathbf{67.89 \pm 0.23}$	52.26 ± 1.14	$\mathbf{74.85 \pm 0.33}$	$\mathbf{88.15 \pm 0.37}$

5.8.1 Datasets

We opted to use many standard computer vision datasets to validate our model's performance. We used the CIFAR-10/100 dataset family for small to medium classes. Each dataset contains 60 000, 32×32 RGB images, with 50 000 images for training and 10 000 for testing. CIFAR-10 has 10 classes with 6000 images per class, and CIFAR-100 has 600 images for each of the 100 classes.

We used the STL-10 unsupervised and supervised sets for fine-tuning and semi-supervised experiments. The unsupervised portion contains 100 000, 96×96 RGB images, and the supervised set has 5000 images from 10 classes.

For datasets with many classes, we used the downsampled version of the original ImageNet dataset, referred to as DS ImageNet. The downsampled version of ImageNet has 1000 classes with 1 281 167, 64×64 RGB images.

5.8.2 Backbones

For all experiments, the encoder $f(\cdot)$ comprises a ResNet-18 backbone followed by a non-linear 2-hidden layer fully-connected network $g(\cdot)$, as a projection head. For CARL, SwAV, and BYOL, the projection head also has batch normalization layers. The functions $g(f(\cdot))$ encode a view v_i into a 128-dim representation z_i. Specifically, the projection head receives a 512-dim vector from the final average pooling layer of the ResNet-18 encoder. The hidden layer of the projection head $g(\cdot)$ contains 512 neurons. For all methods, the dimensionality of the embedding vector z_i and the complexity of both the encoder and projection head are equivalent.

5.8.3 Augmentations

To construct synthetic views for optimizing CARL, we used the same pipeline of data augmentations proposed by Chen et al. [57]. First, we apply a random crop resize operation, which randomly extracts a portion of the image ranging from 0.8 % to 100 % of the original image's area. Second, we apply a horizontal flipping operation with a 50 % chance. There is an 80 % possibility of jittering the pixels of the image, altering its

```
augmentations = transforms.Compose([
    transforms.RandomResizedCrop(image_size, scale=[0.08, 1.0),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomApply(
        [transforms.ColorJitter(brightness=0.8, contrast=0.8,
                                saturation=0.8, hue=0.2)],
        p=0.8
    ),
    transforms.RandomGrayscale(p=0.2),
    GaussianBlur(p=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5],
                         std=[0.5, 0.5, 0.5])
])
```

Listing 1. Code snippet used to generate synthetic views for CARL

brightness, contrast, saturation, and hue. Then, a 20 % chance of a grayscale conversion, a 50 % chance of gaussian blurring, followed by normalization using the dataset's mean and Standard Deviation (STD).

A PyTorch pseudo-code to generate synthetic views for CARL is presented in the code Listing 1.

Using the proposed pipeline for generating synthetic views, CARL is trained with views such as those presented in Figure 5.4.

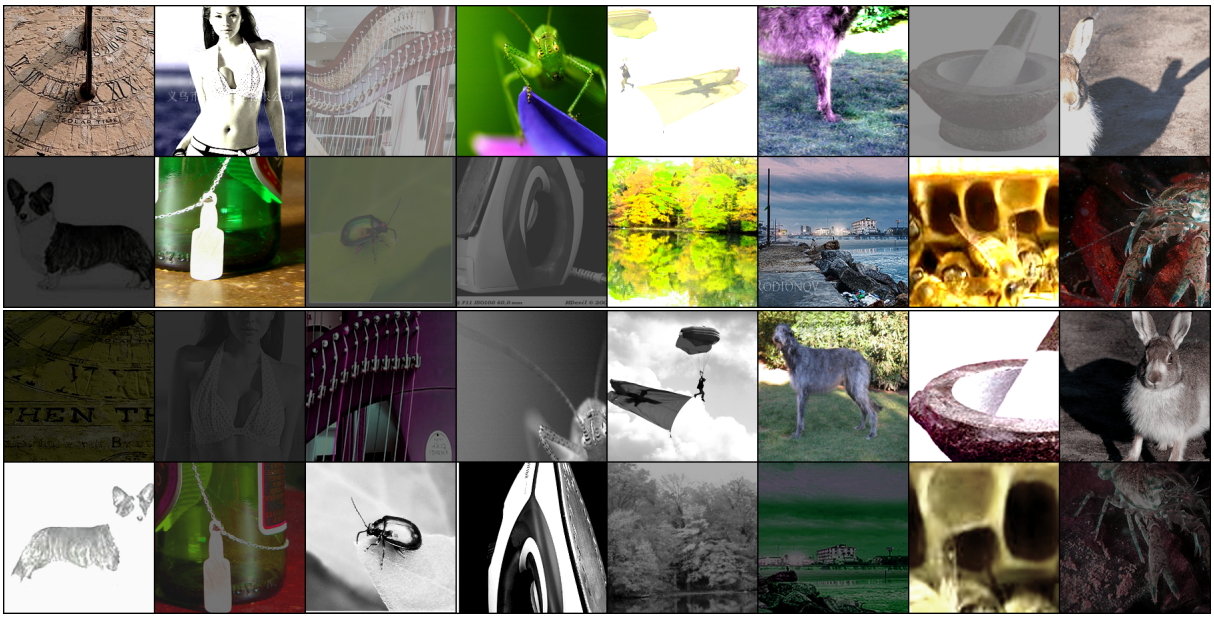

Figure 5.4. Examples of views synthetically created using the pipeline of random transformations used by CARL. The first two rows present transformed versions of 16 images, while the third and fourth rows present different transformations of the same images from above.

5.8.4 CARL

CARL is trained using Stochastic Gradient Descent (SGD) with momentum and a cosine learning rate decay scheduler [109] (without restarts) starting at 0.6 and decaying to 0.0006. We use a weight decay penalty of 5×10^{-4} and the LARS optimizer [110], but using pure SGD produces similar results. To choose the number of general prototypes, we follow the findings from our hyperparameter exploration, see Section 5.6. For CIFAR-10/100, and STL-10, we trained CARL using 100 prototypes for the first dataset and 300 for the last two. Following SwAV, to train in the downsampled ImageNet dataset, we used 3000 prototypes. The projection head of CARL contains batch normalization layers as proposed by Grill et al. [85].

The PyTorch pseudo-code for CARL is shown in code Listing 2.

5.8.5 Other methods

To compare the performance of CARL with other implementations, except for MoCo, we developed our versions of SimCLR, BYOL, SwAV, and PCL by adapting their original code repositories and performing the minimal changes necessary to meet our requirements. The core changes to the official implementations regard (1) adaptation to other datasets, (2) support for single GPU training, (3) change of backbone encoder (from ResNet-50 to ResNet-18), (4) when necessary, change hyperparameters to achieve better results or to comply with our computational budget.

PCL Li et al. [19]. For training on CIFAR-10/100 and STL-10, the number of prototypes is linearly scaled based on the number of true classes. For CIFAR-10 and STL-10, we set the number of clusters to $\{250, 500, 1000\}$, and for CIFAR-100, we used $\{2500, 5000, 10000\}$.

SwAV Caron et al. [87]. For CIFAR-10/100 and STL-10, we used the same number of prototypes used to train CARL. The number of prototypes was set to 100 for CIFAR-10, and 300 for both CIFAR-100 and STL-10. We used the standard configurations from their original repositories to train both PCL and SwAV on the downsampled version of the ImageNet dataset.

BYOL [85] and **SimCLR** [57]. We empirically found that removing the LARS optimizer (from their official implementations) produced better results in our datasets.

We performed a simple grid search to find an optimal learning rate of 0.03, which improved over original parameters.

Originally, BYOL uses a denser representation vector of 256-dim and a more complex projection head with a hidden layer containing 4096 neurons. BYOL employs batch sizes of 4096 observations. All methods use the same projection head architecture, and the extra BYOL's predictor head follows the same architecture as the projection head.

For SimCLR, we empirically found that a temperature parameter of 0.2, instead of 0.5 described in the paper [57], is best for our datasets. We could not reproduce the original training setups for SimCLR and BYOL due to limited hardware. To ensure a fair comparison, we used equal batch sizes of 256 for all implementations. This allows SimCLR to produce 130 560 pairs against 256 pairs used in CARL.

```
def consistency_loss(inputs, targets):
    loss = torch.einsum('nc,nc->n', [inputs, targets])
    loss = - torch.log(loss).mean()
    return loss


def kl_div(p):
    return - torch.sum(p * torch.log(p), dim=1))

# model: convnet + projection head
# assigner: MLP classifier

for x in loader: # load a batch x containing B samples

    # create a pair of synthetic views for each image in the batch
    v1 = aug(x)
    v2 = aug(x)

    z = model(cat(v1, v2)) # embeddings: 2BxD

    scores = assigner(z) # prototype scores: 2BxK

    scores_v1 = scores[:B]
    scores_v2 = scores[B:]

    # convert scores to probabilities
    probs_v1 = Softmax(scores_v1)
    probs_v2 = Softmax(scores_v2)

    consistency = 0.5 * consistency_loss(probs_v1, probs_v2.detach())
    consistency += 0.5 * consistency_loss(probs_v2, probs_v1.detach())

    entropy = kl_div(mean(probs_v1, dim=0))
    entropy += kl_div(mean(probs_v2, dim=0))

    lambda_scale = sample_kl_scalar(train_iter)

    loss = consistency + lambda_scale * entropy

    optimizer.zero_grad()
    loss.backward() # Backward pass
    optimizer.step() # Update the model's parameters
```

Listing 2. Pseudo-PyTorch code for CARL

Supervised. To establish an upper bound for performance comparisons, we trained a supervised ResNet-18 using SGD and a learning rate of 0.03 with a cosine learning rate

decay (without restarts) that decreases to 0. For data augmentation, we used a lighter version of the MoCo's [56] augmentation pipeline composed of (1) random crop resize, (2) random horizontal flip (50 % chance), (3) random color jitter, (4) random gray scaling (20 % chance), and (5) random Gaussian blurring (20 % chance).

5.9 Relation With Other Self-supervised Methods

Here, we compare CARL with other self-supervised methods and further discuss their differences and similarities. We illustrate the differences between existing self-supervised methods and CARL in Figure 5.5.

Relations to SimCLR [57], and MoCo [56]. Different from these models, CARL does not operate directly on the embeddings. Instead, it discretizes the embedding space using general prototypes that serve as energy beacons. These prototypes can attract embeddings from different images that share enough similarity to be drawn by the same prototype. In other words, instead of pushing apart embeddings from different images arbitrarily, CARL indirectly clusters embeddings from different images. MoCo uses an additional memory queue to hold negatives for contrastive training. SimCLR draws negatives from within the batch of size 4096 views. Unlike both, CARL is a non-contrastive method. It does not explicitly push the energy of incompatible pairs up in order to shape the energy function. Instead, by reducing the energy between embeddings and prototypes, the energy between negative embeddings is implicitly pushed up, which frees the method from sampling negative embeddings. On the contrary, both SimCLR and MoCo are contrastive methods and naturally need to mine negatives to shape the energy function correctly. SimCLR uses a siamese architecture while MoCo employs two different encoders, where one of them receives updates in forms of moving averages from the weights of the differential encoder.

Relations to SwAV [87]. CARL shares some similarities with SwAV. Both methods discretize the feature space into a set of finite prototypes. However, to solve the cluster assignment problem and to avoid mode collapse of the joint embeddings training, SwAV uses an offline algorithm from the optimal transport literature called Sinkhorn Knopp, refer to Figure 5.5, d). Consequently, although SwAV leans the prototype vectors online, similarly to CARL, to generate the targets, one needs to run the Sinkhorn Knopp algorithm at each training iteration. On the contrary, CARL learns the general prototypes in a completely differentiable training pipeline. The Sinkhorn Knopp algorithm is an iterative, non-differentiable method and requires memory and tuning of its hyperparameters. In practice, using the Sinkhorn Knopp to solve the clustering problem produces excellent results with the cost of extra computing time. Another difference between CARL and SwAV is that CARL does not require any type of normalization of its embeddings during training. This difference applies to most SSL methods, as can be seen in Figure 5.5.

Relations to BYOL [85], Simple Siamese Networks (SimSiam) [84], and Barlow Twins [90]. Similar to CARL, BYOL, SimSiam and Barlow Twins are non-contrastive methods. SimSiam is very similar to BYOL, but it uses a siamese architecture instead of a momentum-based encoder. BYOL operates a joint embedding, non-symmetric

architecture trained with a simple Mean Squared Error (MSE) loss between embeddings of different views. Additionally to the projection head, both BYOL and SimSiam use a predictor network, refer to Figure 5.5 b) and c) for a visual representation. BYOL adopts the momentum-based encoder; only one of the networks receives gradient updates. The weights of the non-differentiable network are updated with an exponential moving average from the weights of the differentiable network. Both BYOL and SimSiam employ the stop-gradient operation in one of the branches to avoid trivial solutions. Barlow Twins employs a siamese architecture, Figure 5.5 e). Unlike other methods, Barlow Twins avoids collapsed solutions by maximizing the information within the embeddings, which is done by forcing the empirical cross-correlation matrix between the two embeddings to be as close to the identity matrix as possible. Unlike BYOL, CARL does not employ a momentum-based encoder, and different from the others, it does not operate directly on the embeddings.

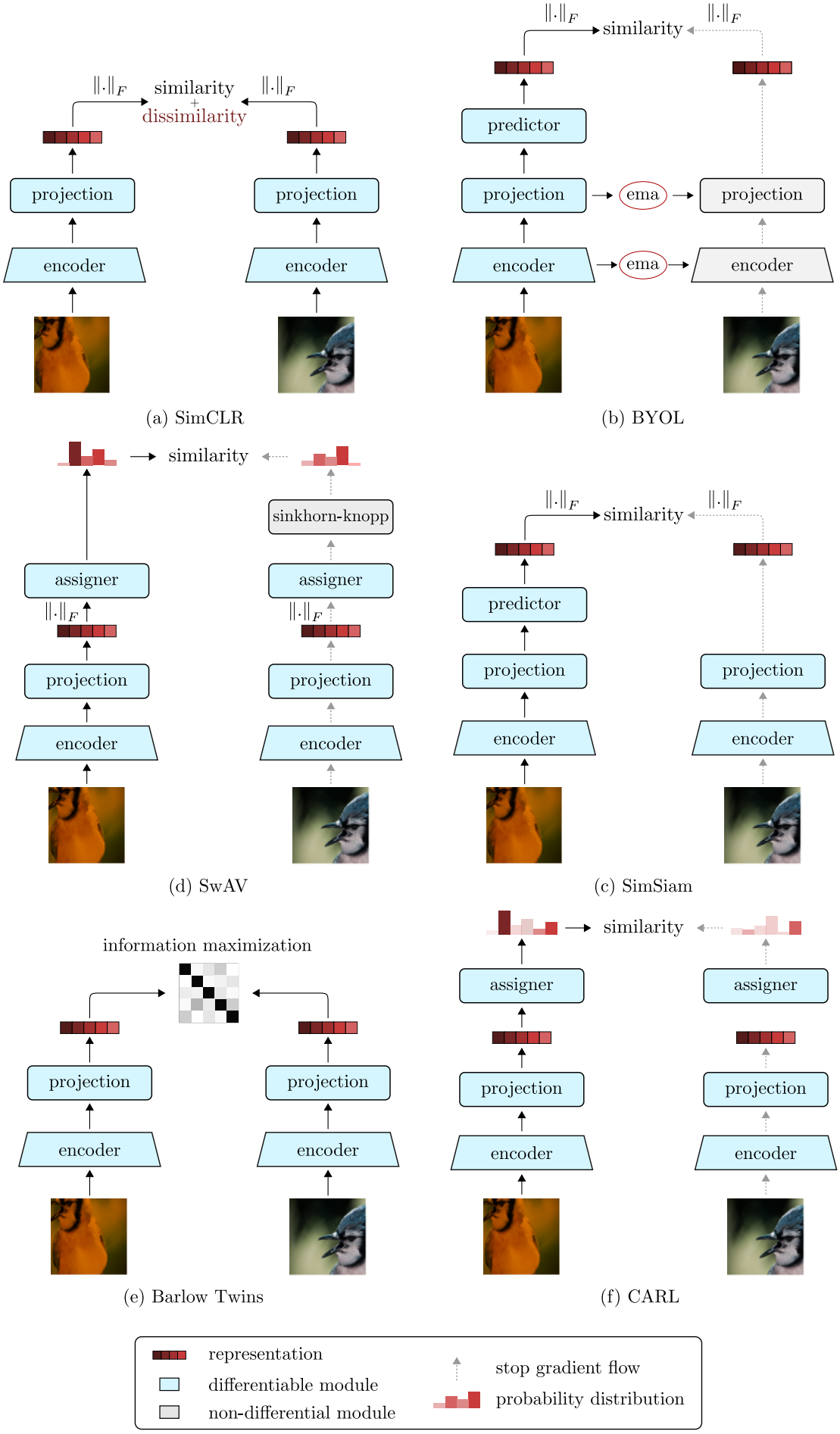

Figure 5.5. Conceptual differences between SSL methods.

Chapter 6

Consistent Assignment of Random Partition Sets

As we have seen in Chapter 5, Consistent Assignment for Representation Learning (CARL) managed to achieve state-of-the-art (SOTA) performance in many representation learning benchmarks. Training joint-embedding architectures mainly differ in how we avoid collapsed solutions during optimization. For CARL, the entropy term in Equation (5.5) is the primary force that avoids training views being assigned to the same prototype. However, as demonstrated in the ablation Section 5.6, if the value of the hyperparameters λ is not properly tuned, CARL will inevitably collapse to a trivial solution, and the learned representations will be useless. In fact, CARL is very sensitive to the choice of hyperparameters such as the value of λ, the batch size, and the number of prototypes C. If these configurations are not correctly balanced, learning representations using CARL can be very unstable. In this chapter, we introduce a solution to these issues. We are going to perform slight changes in the learning protocol described in Section 5.5, and devise a novel pretext task based on the concepts of Set Partitions. We alleviate all the problems described above by working with partitions of sets. This change allows us to model substantially larger probability distributions of general prototypes and, at the same time, reduce the optimized probability distribution size during training. As a result, Consistent Assignment of Random Partition Sets (CARP) can be trained faster, without relying on large batch sizes and being much more stable to hyperparameter changes.

6.1 Motivation

Training an unsupervised system by minimizing the loss in Equation (5.7) can be very unstable. The main limitation is how to avoid trivial solutions. If the decay schedule for λ_e, described in, Subsection (5.6.1) is not tuned correctly, training collapses. For instance, if the value of λ_e is too small, the consistency term wins the arms race, and the average distribution over the batch, Equation (5.6), becomes one-hot alike, i.e., all views end up assigned to the same prototype. If the value of λ_e is too large, the entropy term gets the upper hand, and collapse is avoided. However, the process of view assignment is neglected over the policy of distributing views uniformly, which results in poor performance of the

learned representations. This phenomenon is visually described in Figure 5.2.

For a small number of general prototypes, training is more stable, and the model avoids collapse with a simple tuning of the lambda parameter. However, for a larger number of general prototypes, stability becomes an issue. The main problem lies with the entropy term. For more interesting cases, when the distribution is larger, regular batch sizes, such as 64 or 128, become too small to properly model the distribution, the signal is too weak for most prototypes. Consequently, to avoid collapsing, we need to increase the contribution of the entropy term or increase the batch size.

Another consequence of modeling large distributions of general prototypes with moderate to small batch sizes regards distributed training. As usually implemented in most deep learning frameworks, distributed training usually evenly splits the dataset and the batch size over a number of parallel nodes. Unfortunately, splitting the batch size causes individual nodes to use even smaller batch sizes, which increases the instability of the entropy term, leading to mode collapse.

To address such limitation, we propose to decouple the loss function in Equation (5.7) into smaller sub-problems. Instead of enforcing both consistency and uniform assignments over all the general prototypes, we propose an optimization task over subsets or blocks of a random partition of the general prototypes set C.

6.2 Towards Random Partition Sets

Similar to other Self-Supervised Learning (SSL) algorithms, CARP operates in a siamese join-embeddings architecture. The setup is very similar to CARL. From an image x, we create two views of x, denoted as $v_i = T(x)$ and $v_j = T(x)$ using a stochastic function T that applies a set of random image transformations. The two views are forwarded through the siamese encoder, f_θ, to produce respective embedding vectors $z_i = f_\theta(x)$ and $z_j = f_\theta(x)$.

Following clustering-based ideas, we discretize the embedding space into a set of general prototype vectors. Note that these prototypes are not meant to represent the true classes of the data; instead, our general prototypes can be viewed as anchors to attract views of a given image to a commonplace in the embedding space. A function q_θ, implemented as a neural network, receives the representation vectors as input and outputs a vector of energy values relating the views' embeddings with the prototypes. Different from CARL the assigner module q_θ, is a two-layer non-linear Multilayer Perceptron (MLP).

CARL's loss function is composed of two terms: the consistency and the entropy terms. The consistency term learns the relations between embedding vectors and prototypes; refer to Equation (5.4). For a pair of views, the consistency loss is optimized when the two views are assigned to the same prototype with maximal confidence. Moreover, the entropy term, Equation (5.5) can be viewed as a regularizer. It prevents the system from collapsing by ensuring that, for a batch of size B, on average, each prototype roughly receives the same number of assignments. Since we want to maximize the entropy over assignments averaged within a batch, we can view the entropy term as enforcing a uniform distribution relating embeddings to prototypes.

Now, we introduce Partitions of sets into CARL's learning framework.

6.3 From CARL to CARP

Given the set of K prototypes $C = \{c_1, c_2, ..., c_K\}$, we define a partition of C as $P = \{P_i\}_i$, such that $\emptyset \notin P$, $\bigcup_i b_i = C$ where $b_i \in P$, and $b_i \cap b_j = \emptyset$ where $\forall b_i, b_j \in P$, and $i \neq j$. We are interested in the partitions $P = \{P_i\}_i$ of size N_P, i.e., $|P_i| = N_P$.

Using the concept of a partition of a set, we can define a framework of pretext tasks over partition blocks that satisfies the learning problem defined in Section 5.5. If the size of a partition block equals the number of prototype, $N_B = K$, then the partition P contains only one block which is equivalent to

$$P = \{b_0\} = \{C\}\,. \tag{6.1}$$

If the size of the partition blocks equals $N_B = 1$, then we have K blocks in the partition P, and each block has a unique prototype. Here, the learning task is equivalent to a binary classification problem, where each output score, if normalized, expresses the likelihood of a data point x_i to independently belong to each prototype.

However, if the partition's block size $1 < N_B < K$, and N_B divides K, then the partition P will be composed of blocks $B = \{b_0, b_1, ..., b_{N_P}\}$, where $N_P = \lfloor K/N_B \rfloor$ is the number of blocks, and P is defined as:

$$P = \{\{b_0\}, \{b_1\}, ..., \{b_{N_P}\}\} = \left\{ \underbrace{\{c_i, c_j, ..., c_v\}}_{\text{size } N_B}, ..., \underbrace{\{c_k, c_q, ...c_r\}}_{\text{size } N_B} \right\}, \tag{6.2}$$

where indices i, j, v, k, q, and r are unique and randomly chosen.

Instead of mapping the representations to assignments as a linear combination of all prototypes and the views' representations, Equation (5.2), we compare the representation z_i, for a view i against all the prototypes in a block j, as

$$q_{i,j} = \langle z_i, b_j \rangle\,, \tag{6.3}$$

to obtain an energy distribution w.r.t. the prototypes of a particular block index by j. Figure 6.1 (a), depicts CARP's learning framework and compares it with CARL.

Here, $q_{i,j}$ is the combination of the representation from the i-th view with the prototypes from the j-th block of the random partition.

Next, to get a distribution of a view over the prototypes of a block, we normalize the energy using the softmax function and obtain the posterior probability distribution, i.e., the probability of assigning the view v_i to the prototypes of a block b_j. Hence, our normalized probability for a view v_i to belong to the k-th class among the prototypes within a block b_j is

$$p_{i,j}[k] = P(i \text{ assigned to } k \mid v_i) = \frac{\exp(q_{i,j}[k])}{\sum_{m=1}^{K} \exp(q_{i,j}[m])}. \tag{6.4}$$

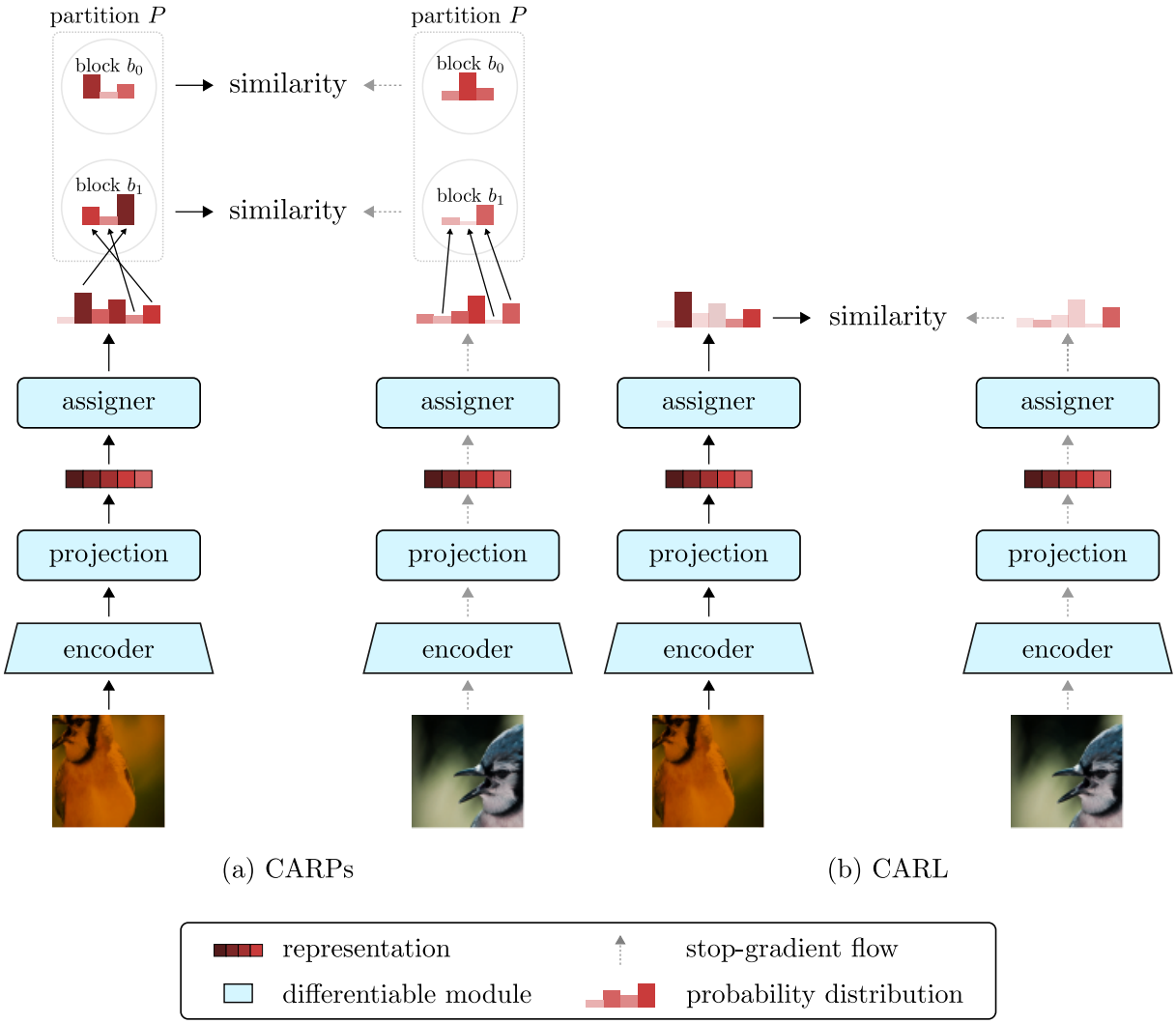

Figure 6.1. Conceptual differences between CARL and CARP. The encoding process is similar. A pair of views v_i and v_j are passed through a siamese encoder and projection head to produce respective low-level representation vectors. The representation vectors are combined with a set of prototypes C, and consistent assignment of views is enforced. Instead of modeling the entire probability distribution of views over prototypes, CARP randomly partition the set of prototypes C. A partition P is composed of N_P blocks, and each block contains an equal number of prototypes N_B. The learning task is equivalent to CARL's but done block-wise.

To ensure the similarity between the views, we optimize the views' distributions $p^a_{i,j}$ and $p^+_{i,j}$ over the prototypes of a block indexed by j, so that the two distributions are consistent with one another. Note that superscripts a and $+$ denote anchor and positive views. We use the superscript $+$, instead of p to refer to a positive view to avoid confusion since p is here defined as a probability distribution.

The consistency term of our partition consistency loss is now defined as,

$$\mathcal{L}_c = -\frac{1}{BN_P}\sum_{i}^{B}\sum_{j}^{N_P}\log\left\langle p^a_{i,j}, p^+_{i,j}\right\rangle. \tag{6.5}$$

Following similar reasoning, the block-wise entropy term is defined as,

$$\mathcal{L}_{\mathrm{KL}} = \mathrm{KL}(\mathcal{P}_j \parallel U) = \log(K) + \sum_{j}^{N_P}\sum_{c\in C}\hat{p}^c_j \log\left(\hat{p}^c_j\right), \tag{6.6}$$

where $\mathcal{P}_j$ is the distribution over the prototypes of block j, and $\hat{p}^c_j$ is the expected distribution over a minibatch of size B for a block j,

$$\hat{p}^c_j = \frac{1}{B}\sum_{i}^{B} p^c_{i,j}, \tag{6.7}$$

and $p^c_{i,j}$ is the probability of assigning a view v_i to a cluster $c \in C$ that belongs to a given partition block b_j.

Note that Equations (5.4) and (5.5) from CARL are very similar to Equations (6.5) and (6.6) recently introduced. The main difference is that instead of dealing with the entire probability distribution at once, they perform optimization block by block.

While the $\mathrm{KL}(\mathcal{P} \parallel U)$ in Equation (5.5) forces the entire distribution over prototypes C to be uniform, the proposed block-wise KL, Equation (6.6), forces uniform assignment of views over a subset of random prototypes within partition blocks. Since the uniform distribution over partition block b_j is equivalent to the uniform distribution of the entire set of prototypes, the regularizer in Equation (6.6) has the same effect as Equation 5.5, with the advantage of optimizing over a small set of prototypes.

Combining partition consistency with entropy regularization, and adding the trade off hyperparameter λ_e, the final objective is defined as:

$$\mathcal{L} = -\frac{1}{BN_P}\sum_{j}^{N_P}\sum_{i}^{B}\log\left\langle p^a_{i,j}, p^+_{i,j}\right\rangle + \lambda_e\left(\log(K) + \sum_{c\in C}\hat{p}^c_j\log\left(\hat{p}^c_j\right)\right), \tag{6.8}$$

which can be simplified to

$$\mathcal{L} = -\frac{1}{BN_P}\sum_{j}^{N_P}\mathcal{L}_{\mathrm{c}} + \lambda_e\mathcal{L}_{\mathrm{KL}}. \tag{6.9}$$

Table 6.1. Image classification with linear models. We report top-1 accuracy. Numbers are adopted from corresponding papers. †: A Simple Framework for Contrastive Learning of Visual Representations (SimCLR), Bootstrap Your Own Latent (BYOL), and Swapping Assignments between Views (SwAV) use a large batch size of 4096. †: SwAV uses multi-crop augmentation and extra queue containing 3840 embeddings.

Method	Epochs	Top-1	Top-5
Supervised	100	76.5	-
SimCLR	200	61.9	
MoCo v2	200	67.7	-
PCL v2	200	67.6	-
CARL (**ours**)	200	65.7	86.7
CARP (**ours**)	200	**68.6**	**88.5**
results of longer unsupervised training			
PIRL	800	63.6	-
SimCLR	1000	69.3†	89.0†
MoCo v2	800	71.1	**90.1**
SimSiam	800	71.3	-
SwAV (w/o multicrop)	400	71.8	-
BYOL	1000	74.3†	71.6†
SwAV	800	**75.3**†	-
CARP (**ours**)	300	70.1	89.4

6.4 Unsupervised Feature Evaluation

To measure the quality of the representations learned by CARP, we report performance using the linear evaluation and the semi-supervised protocols on the ImageNet-1k dataset. We compare the performance of CARP against the SOTA methods on unsupervised visual representations learning. We report results from pre-training CARP on the unsupervised ImageNet-1k dataset for 200 and 300 epochs.

6.4.1 Linear Evaluation

Table 6.1 shows the performance of training a linear classifier on top of the fixed representations learned by CARP. After pre-training, we learn a linear model on top of the frozen representations from the ResNet-50 encoder for 100 epochs on ImageNet-1k labeled data. The evaluation protocol follows Subsection 4.2.1. Among the methods pre-trained for 200 epochs, CARP achieves the best top-1 and top-5 accuracy. In the ImageNet dataset, CARP beats CARL by 3.9 extra points of accuracy. For more extended training regimes, CARP achieves 70.1 accuracy, which puts it on pair with other methods.

6.4.2 Semi-supervised Learning

Table 6.2 reports performance on the semi-supervised learning evaluation protocol described in Subsection 4.2.2. After pre-training, we fine-tune the ResNet-50 encoder for 20 epochs using different portions (1% and 10%) of the ImageNet-1k labeled data. Among

Table 6.2. Semi-supervised learning on ImageNet. We report top-5 accuracy on the ImageNet validation set of self-supervised models that are finetuned on 1% or 10% of labeled data. †: SimCLR, BYOL, and SwAV use a large batch size of 4096. †: SwAV uses multi-crop augmentation and extra queue containing 3840 embeddings. †: Barlow Twins uses a more complex projection head, batch sizes of 2048 and 8192-dim embeddings.

Method	Architecture	Epochs	Top-5 accuracy	
			1%	10%
Supervised	RN50	-	48.4	80.4
SimCLR	RN50	200	56.5	82.7
MoCo v2	RN50	200	66.3	84.4
PCL v2	RN50	200	73.9	85.0
CARL (**ours**)	RN50	200	72.6	83.9
CARP (**ours**)	RN50	200	**74.5**	**85.3**
results of longer unsupervised training				
BYOL	RN50	1000	78.4†	89.0†
SwAV	RN50	800	78.5†	**89.9**†
PIRL	RN50	800	57.2	83.8
Barlow Twins	RN50	1000	**79.2**†	89.3†
SimCLR	RN50	1000	75.5†	87.8†
CARP (**ours**)	RN50	300	77.1	86.7

self-supervised methods pre-trained for 200 epochs, CARP achieved the best overall top-5 accuracy using 1% and 10% of labels. Among methods pre-trained for more than 200 epochs, CARP achieved 77.1 and 86.7 top-5 accuracy when using 1% and 10% of labeled data, respectively. Note that SwAV, which was trained for 800 epochs, accomplished very similar numbers.

6.5 Implementation Details

6.5.1 Datasets

In order to compare the performance of CARP against SOTA SSL methods, we use the ImageNet-1k dataset. For more details, please refer to Section 4.1.

6.5.2 Backbones

For all experiments, the encoder $f(\cdot)$ comprises a ResNet-50 Convolutional Neural Networks (CNN) backbone followed by a non-linear 2-hidden layer MLP $g(\cdot)$ denoted as the projection head. The functions $g(f(\cdot))$ encode a view v_i into a 128-dim representation z_i. Specifically, the projection head receives a 2048-dim vector from the final average pooling layer of the ResNet-50 encoder. The hidden layer of the projection head $g(\cdot)$ contains 2048 neurons. The assigner function $q(\cdot)$ receives the low-dimensional representations vector z_i as input and outputs an energy vector relating the views with the prototypes in C. The functions $q(\cdot)$ and $g(\cdot)$ have the same number of neurons and both contains batch

normalization layers.

6.5.3 Augmentations

To produce synthetic views for optimizing CARP, we used the same pipeline of data augmentations proposed by Grill et al. [85]. The pipeline for random transformations is described below:

- Random crop resize operation: Extracts a portion of the original image, ranging from 12.5 % to 92.5 % of the image's size, then resize the crop to $224 \times 224 \times 3$,
- Random horizontal flipping: 50 % chance,
- Pixel jittering: 80 % chance of jittering the pixels of an image, altering its brightness, contrast, saturation, and hue,
- Random grayscale conversion: 20 % chance,
- Random gaussian blurring: 50 % chance,
- Random solarization: 20 % chance for the second view.

After, we normalize the resulting image using the dataset's mean and Standard Deviation (STD).

A PyTorch pseudo-code to generate synthetic views for CARP is presented in the code Listing 3.

6.5.4 CARP

CARP is trained with the LARS optimizer [110] using 4 Graphics Processing Unit (GPUs) and a total batch size of 256 images. The learning rate decay schedule follows a cosine decay with no warm-ups or restarts; it is set to 5.4 and decays to 0.00054 throughout training. All the learning modules have a weight decay penalty of 1×10^{-6}. The random partition size is defined as $N_P = 12$ which yields individual block sizes of $N_B = 250$ random prototypes. Note that we reinitialize the partitions at each iteration. Lastly, the KL weight term λ_e starts as $b = 2.0$ and decays to $a = 0.15$ during the first 30 epochs of training.

```
# To create view #1
self.transform = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=[0.15, 0.925]),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomApply(
        [transforms.ColorJitter(brightness=0.8, contrast=0.8,
                                saturation=0.8, hue=0.2)],
        p=0.8
    ),
    transforms.RandomGrayscale(p=0.2),
    GaussianBlur(p=1.0),
    Solarization(p=0.0),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# To create view #2
self.transform_prime = transforms.Compose([
    transforms.RandomResizedCrop(224, scale=[0.15, 0.925]),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomApply(
        [transforms.ColorJitter(brightness=0.8, contrast=0.8,
                                saturation=0.8, hue=0.2)],
        p=0.8
    ),
    transforms.RandomGrayscale(p=0.2),
    GaussianBlur(p=0.1),
    Solarization(p=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

Listing 3. Code snippet used to generate synthetic views for CARP. Note that the process of view generation slightly differs for views one and two. For view number 1, we do not apply solarization, while for view number 2, we apply solarization with a 20 % chance.

Chapter 7

Conclusions

This dissertation presented a comprehensive overview of the existing Self-Supervised Learning (SSL) methods for Computer Vision (CV). Most importantly, it also introduced two novel algorithmic approaches to learn representations from unlabeled images, namely Consistent Assignment for Representation Learning (CARL) and Consistent Assignment of Random Partition Sets (CARP). Experiments for CARL and CARP revealed that the two methods are effective in learning representations from unlabeled images. The representations learned by CARL and CARP demonstrated useful generalization properties when utilized to learn new downstream tasks. Both methods employ a novel strategy to avoid trivial solutions in joint-embedding architectures and are end-to-end systems based on deep clustering and SSL. The results of our methods are comparable with contemporary state-of-the-art (SOTA) methods in representation learning with advantages such as less computing time. Although our methods were introduced and trained on classic vision datasets such as the ImageNet-1k and the CIFAR family, the techniques we proposed are general enough to be used on any other vision modality, including medical or geospatial imagery or different data modalities such as text or audio signals.

7.1 Future Work

SSL is interested in learning models of the world in a task-independent label-free way. Once the model has a good understanding of general concepts, which translates to generalizable representations, the pre-trained model can be used to learn new downstream tasks without relying upon extensive quantities of labeled examples. An auspicious direction of SSL for CV is to break out from the imaging modality and use videos to learn representations. Today, the most successful SSL methods learn representations from individual images with no time dependency. Moreover, as we discussed in Subsection 3.2.1, to simulate different viewpoints of an object, we rely on extensive data augmentation, which is widely considered a manual process susceptible to human biases. Ideally, using videos instead of images could allow us to exchange the ad-hoc data augmentations to a hierarchical framework incorporating time dependencies between different concepts. In fact, video data already incorporates the characteristics of the instance discrimination pretext task, and it is a widely available source of information. If we want to learn hierarchical

representations by observing different objects from different viewpoints, video data seem to be a natural candidate. However, learning representations from video frames is not trivial due to the high dimensional spaces in which image data live. In fact, recent work by Baevski et al. [111] has attempted to learn hierarchical representations from video data, and their results suggest that video data is a promising source of information for unsupervised representation learning methods.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[2] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[3] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[7] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[8] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[9] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with exemplar convolutional neural networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 9, pp. 1734–1747, 2015.

[12] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2051–2060.

[13] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[14] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1735–1742.

[15] 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.

[16] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 132–149.

[17] M. Caron, P. Bojanowski, J. Mairal, and A. Joulin, "Unsupervised pre-training of image features on non-curated data," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2959–2968.

[18] X. Yan, I. Misra, A. Gupta, D. Ghadiyaram, and D. Mahajan, "Clusterfit: Improving generalization of visual representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6509–6518.

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

[20] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European conference on computer vision*. Springer, 2016, pp. 649–666.

[21] B. Marr, "How much data do we create every day? the mind-blowing stats everyone should read," in *Forbes*, "2018 (accessed July 10, 2020)". [Online]. Available: "https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#75335d3d60ba"

[22] M. Blaschko, P. Kumar, and B. Taskar, "Tutorial: Visual learning with weak supervision," in *CVPR*, "2013 (accessed July 10, 2020)". [Online]. Available: "http://mpawankumar.info/tutorials/cvpr2013/index.html"

[23] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[24] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[28] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[29] G. Litjens, P. Bandi, B. Ehteshami Bejnordi, O. Geessink, M. Balkenhol, P. Bult, A. Halilovic, M. Hermsen, R. van de Loo, R. Vogels *et al.*, "1399 h&e-stained sentinel lymph node sections of breast cancer patients: the camelyon dataset," *GigaScience*, vol. 7, no. 6, p. giy065, 2018.

[30] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[31] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[32] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," *Advances in neural information processing systems*, vol. 26, pp. 2292–2300, 2013.

[33] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[34] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[35] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.

[36] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International Conference on Machine Learning*. PMLR, 2016, pp. 1747–1756.

[37] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," *arXiv preprint arXiv:1606.05328*, 2016.

[38] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *International Conference on Machine Learning*. PMLR, 2020, pp. 1691–1703.

[39] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[40] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2536–2544.

[41] T. Malisiewicz, A. Gupta, and A. A. Efros, "Ensemble of exemplar-svms for object detection and beyond," in *ICCV*, 2011.

[42] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 215–223.

[43] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Journal of Software Engineering and Applications, Vol.11 No.2, February 6, 2018*, 2009.

[44] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *CVPR*. IEEE, 2004, pp. 178–178.

[45] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[46] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[47] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1422–1430.

[48] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[49] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European Conference on Computer Vision*. Springer, 2016, pp. 69–84.

[50] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *arXiv preprint arXiv:1803.07728*, 2018.

[51] R. Arandjelovic and A. Zisserman, "Objects that sound," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 435–451.

[52] B. Korbar, D. Tran, and L. Torresani, "Cooperative learning of audio and video models from self-supervised synchronization," in *Advances in Neural Information Processing Systems*, 2018, pp. 7763–7774.

[53] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," vol. 33, 2020, pp. 12 449–12 460.

[54] I. Misra and L. v. d. Maaten, "Self-supervised learning of pretext-invariant representations," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 6707–6717.

[55] A. Kolesnikov, X. Zhai, and L. Beyer, "Revisiting self-supervised visual representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019, pp. 1920–1929.

[56] 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*, 2020, pp. 9729–9738.

[57] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," *arXiv preprint arXiv:2002.05709*, 2020.

[58] A. Van Den Oord, O. Vinyals *et al.*, "Neural discrete representation learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 6306–6315.

[59] J. Donahue and K. Simonyan, "Large scale adversarial representation learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 10 542–10 552.

[60] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," *Predicting structured data*, vol. 1, no. 0, 2006.

[61] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv preprint arXiv:1406.2661*, 2014.

[62] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," *arXiv preprint arXiv:1609.03126*, 2016.

[63] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis," *Advances in neural information processing systems*, vol. 17, 2004.

[64] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[65] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 04, pp. 669–688, 1993.

[66] 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*, 2014, pp. 1386–1393.

[67] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," in *Advances in neural information processing systems*, 2016, pp. 1857–1865.

[68] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus, "Hard negative mixing for contrastive learning," vol. 33, 2020, pp. 21 798–21 809.

[69] J. Robinson, C.-Y. Chuang, S. Sra, and S. Jegelka, "Contrastive learning with hard negative samples," in *Inter. Conf. Learn. Represent. Wksps. (ICLRW)*, 2020.

[70] 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*, 2018, pp. 3733–3742.

[71] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," in *Wksp. Adv. Neural Inf. Process. Sys. (NeurIPSW)*, 2018.

[72] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[73] O. J. Hénaff, A. Srinivas, J. De Fauw, A. Razavi, C. Doersch, S. Eslami, and A. v. d. Oord, "Data-efficient image recognition with contrastive predictive coding," *arXiv preprint arXiv:1905.09272*, 2019.

[74] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," *arXiv preprint arXiv:1906.05849*, 2019.

[75] X. Chen, H. Fan, R. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *arXiv preprint arXiv:2003.04297*, 2020.

[76] 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.

[77] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[78] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton, "Big self-supervised models are strong semi-supervised learners," in *Wksp. Adv. Neural Inf. Process. Sys. (NeurIPSW)*, 2020.

[79] P. Bachman, R. D. Hjelm, and W. Buchwalter, "Learning representations by maximizing mutual information across views," in *Advances in Neural Information Processing Systems*, 2019, pp. 15 535–15 545.

[80] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning," *arXiv preprint arXiv:2005.10243*, 2020.

[81] Y. Tian, D. Krishnan, and P. Isola, "Contrastive representation distillation," *arXiv preprint arXiv:1910.10699*, 2019.

[82] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[83] Y. Tian, X. Chen, and S. Ganguli, "Understanding self-supervised learning dynamics without contrastive pairs," in *Inter. Conf. Mach. Learn. (ICML)*. PMLR, 2021, pp. 10 268–10 278.

[84] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 750–15 758.

[85] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar *et al.*, "Bootstrap your own latent: A new approach to self-supervised learning," *arXiv preprint arXiv:2006.07733*, 2020.

[86] Y. M. Asano, C. Rupprecht, and A. Vedaldi, "Self-labelling via simultaneous clustering and representation learning," *arXiv preprint arXiv:1911.05371*, 2019.

[87] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *arXiv preprint arXiv:2006.09882*, 2020.

[88] P. Goyal, M. Caron, B. Lefaudeux, M. Xu, P. Wang, V. Pai, M. Singh, V. Liptchinsky, I. Misra, A. Joulin *et al.*, "Self-supervised pretraining of visual features in the wild," 2021.

[89] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," 2020, pp. 10 428–10 436.

[90] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny, "Barlow twins: Self-supervised learning via redundancy reduction," in *Inter. Conf. Learn. Represent. Wksps. (ICLRW)*, 2021.

[91] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer, "S4l: Self-supervised semi-supervised learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1476–1485.

[92] S. Gidaris, A. Bursuc, G. Puy, N. Komodakis, M. Cord, and P. Pérez, "Online bag-of-visual-words generation for unsupervised representation learning," *CVPR*, 2020.

[93] F. C. Ghesu, B. Georgescu, A. Mansoor, Y. Yoo, D. Neumann, P. Patel, R. Vishwanath, J. M. Balter, Y. Cao, S. Grbic *et al.*, "Self-supervised learning from 100 million medical images," *arXiv preprint arXiv:2201.01283*, 2022.

[94] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," vol. 30, 2017.

[95] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9650–9660.

[96] S. Mo, H. Kang, K. Sohn, C.-L. Li, and J. Shin, "Object-aware contrastive learning for debiased scene representation," vol. 34, 2021.

[97] S. Purushwalkam and A. Gupta, "Demystifying contrastive self-supervised learning: Invariances, augmentations and dataset biases," vol. 33, 2020, pp. 3407–3418.

[98] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *European conference on computer vision*. Springer, 2014, pp. 446–461.

[99] T. Berg, J. Liu, S. Woo Lee, M. L. Alexander, D. W. Jacobs, and P. N. Belhumeur, "Birdsnap: Large-scale fine-grained visual categorization of birds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2011–2018.

[100] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *Proceedings of the IEEE international conference on computer vision workshops*, 2013, pp. 554–561.

[101] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge 2007 (voc2007) results," 2007.

[102] T. S. Silva and A. Ramírez Rivera, "Consistent assignment for representation learning," in *Energy Based Models Workshop-ICLR 2021*, 2021.

[103] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," *Adv. Neural Inf. Process. Sys. (NeurIPS)*, pp. 737–737, 1994.

[104] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[105] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[106] W. Van Gansbeke, S. Vandenhende, S. Georgoulis, M. Proesmans, and L. Van Gool, "Learning to classify images without labels," *arXiv preprint arXiv:2005.12320*, 2020.

[107] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training imagenet in 1 hour," in *IEEE Inter. Conf. Comput. Vis. Wksps. (ICCVW)*, 2018.

[108] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the CIFAR datasets," *arXiv preprint 1707.08819*, 2017.

[109] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Inter. Conf. Learn. Represent. Wksps. (ICLRW)*, 2016.

[110] Y. You, I. Gitman, and B. Ginsburg, "Large batch training of convolutional networks," *arXiv preprint 1708.03888*, 2017.

[111] A. Baevski, W.-N. Hsu, Q. Xu, A. Babu, J. Gu, and M. Auli, "data2vec: A general framework for self-supervised learning in speech, vision and language," *arXiv preprint arXiv:2202.03555*, 2022.