

https://dspace.org/

A Novel Data Augmentation Framework for Contrastive Self-Supervised Learning

Fokam, Manuel Arnol

2024-09
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/50268
Fokam, Manuel Arnol. (2024). A Novel Data Augmentation Framework for Contrastive Self-Supervised Learning. [Master's dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace. https://hdl.handle.net/10539/50268

Downloaded from a DSpace repository. https://dspace.org/

A Novel Data Augmentation Framework for Contrastive Self-Supervised Learning

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Manuel Fokam
2490967

Supervised by Prof. Richard Klein

September 3, 2024



A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science.

Declaration

I, Manuel Arnol Fokam, hereby declare that this dissertation is my own work. I submit this dissertation for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university or for any other degree.



September 3, 2024

Abstract

Self-supervised learning (SSL) has emerged as a leading paradigm in visual representation learning, employing techniques that encourage similarity among latent representations produced from diverse transformations of the same image. Despite the proven impact of data augmentation on such methods, the design of a suitable data augmentation pipeline for training such methods disregards the downstream tasks for which the learned representation is intended. To overcome this challenge, we introduce a novel data augmentation framework called Rank-Update (RU).

This framework periodically assesses the relevance of different augmentation techniques for various downstream tasks, enabling the construction of an augmentation pipeline tailored to the specific requirements of those tasks without sacrificing generalization. This approach is appropriate in cases where we know the downstream tasks in advance but do not have enough data to train a model on these tasks directly through supervised learning. We first present an empirical study on the influence of different data augmentation techniques on SSL models across various downstream tasks. Furthermore, we then demonstrate the effectiveness of our approach in comparison to other methods on the same set of downstream tasks. Finally, we showcase the robustness of our method across different configurations, while presenting its limitations.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Richard Klein, for his invaluable advice and guidance throughout this research project. I am thankful to all my peers at the RAIL and PRIME Lab for the help they provided in various forms during this journey. Most importantly, I extend my heartfelt thanks to my family and friends for their unwavering emotional support and encouragement.

Contents

Preface
Declaration i
Abstract ii
Acknowledgements iii
Table of Contents iv
List of Figures vi
List of Tables viii

1 Introduction **1**

2 Background **4**
2.1 Self-Supervised Learning (SSL) 4
2.1.1 The SSL Formulation 5
2.1.2 SSL in Practice 5
2.2 Contrastive SSL 8
2.2.1 With Negatives 9
2.2.2 Without Negatives 10
2.3 Data-Augmentation in Contrastive SSL 10
2.4 Summary 11

3 Related Work **12**
3.1 Ignoring Data Augmentation 12
3.1.1 Generative SSL 14
3.1.2 Improved Objective Function 15
3.2 Improving Data Augmentation 16
3.2.1 Learnable Augmentations 16
3.2.2 Non-Learnable Augmentations 17
3.3 Summary 18

4 Methodology **19**
4.1 Main Motivation 19
4.2 Modifying Data Augmentation in the Contrastive SSL 19
4.3 Rank-Update Data Augmentation Framework 20
4.3.1 Ranking Mechanism 22
4.3.2 Update Mechanism 26
4.4 Summary 28

5 Experimental Setup **29**
5.1 Experimental Procedure . . . 29
5.2 Pre-Training . . . 29
5.2.1 Pre-Training Dataset . . . 30
5.2.2 Contrastive SSL Methods . . . 30
5.3 Baseline Augmentation Methods . . . 31
5.3.1 Fixed Augmentation Pipeline . . . 32
5.3.2 RandAugment . . . 32
5.3.3 TrivialAugment . . . 33
5.4 Evaluation on Downstream Tasks . . . 33
5.4.1 Evaluation Datasets . . . 33
5.4.2 Evaluation Pipeline . . . 34
5.5 Summary . . . 34

6 Results & Analysis **35**
6.1 The Augmentation Pipeline in Contrastive SSL . . . 36
6.1.1 Choice of the Augmentation Operations . . . 36
6.1.2 Order of the Augmentation Operations . . . 38
6.2 The Rank-Update Framework Against Other Augmentation Methods . . . 39
6.2.1 Performance Against a Fixed Augmentation Pipeline . . . 40
6.2.2 Performance Against Other Parameter-Free Methods . . . 45
6.3 Ablations over Different configurations . . . 45
6.3.1 Rank-Update configurations . . . 45
6.3.2 Ablations over SSL configurations . . . 50
6.4 Summary . . . 54

7 Conclusion & Future Work **56**

References **57**

A Appendix **67**
A.1 Datasets Details . . . 67
A.2 Implementation Details & Hyperparameters . . . 69
A.2.1 Pre-training . . . 69
A.2.2 Linear Evaluation . . . 70
A.2.3 Data Agumentation . . . 70

List of Figures

2.1 The Self-Supervised Learning pipeline. A neural network is trained on an unlabeled dataset through self-supervised pre-training (neural network at the top). Then, the trained network is used to perform different downstream tasks through the updated supervised objective in Equation 2.2 (copies of the network at the bottom). 6

2.2 A contrastive SSL method where a neural network is trained to bring together the representations of views generated from the same data point while pushing apart the representations of views generated from different data points. 8

3.1 Roadmap of the related work section leading to our proposed approach . 13

4.1 Our Rank-Update Framework in the Contrastive SSL pre-training pipeline. 21

6.1 Average performance of SSL algorithms trained with four augmentation operations separately and evaluated on all six downstream tasks. For each bar plot, we normalize the results of the 4 operations with respect to the best performance in that plot to focus on the trends along each downstream task. 37

6.2 The performance of SimCLR with linear evaluation on various downstream tasks is shown here for different 2-stage data augmentation pipelines. In each heatmap, rows represent the first augmentation operation applied, and columns represent the second. The opacity of a cell's color is directly proportional to the average accuracy achieved on the corresponding downstream task 38

6.3 Evolution of the pre-training loss for Barlow Twins and SimCLR with a fixed augmentation pipeline compared to our method. 41

6.4 Evolution of the KNN accuracy for Barlow Twins with our method over different downstream tasks. 42

6.5 Stacked area plot of the evolution probabilities of different operations in the augmentation bag $\mathcal{T}$ throughout pre-training. 42

6.6 Evolution of the pre-training loss for Barlow Twins and SimCLR with a fixed augmentation pipeline compared to our method. 44

6.7 Evolution of the KNN accuracy for Barlow Twins and SimCLR with our method over different downstream tasks. 44

6.8 Stacked area plot of the evolution of probabilities for different operations in the augmentation bag $\mathcal{T}$ throughout pre-training. 44
6.9 Bar plots of top-1 accuracies of SimCLR and Barlow with TrivialAugment, RandAugment and our RU augmentation method. 45
6.10 Average performance of RU on SimCLR at various samples sizes for different downstream tasks. 46
6.11 Average performance of RU on SimCLR at various pipeline sizes for different downstream tasks. 47
6.12 Average performance of RU (with uniform ranker) on SimCLR at various pipeline sizes for different downstream tasks. 47
6.13 Average performance of RU on SimCLR at various pre-training epochs intervals for different downstream tasks. 48
6.14 A heatmap illustrating the performance of RU with SimCLR for various combinations of ranking and evaluation downstream tasks. The rows correspond to the downstream task used for ranking, while the columns represent the downstream tasks used for evaluating the model. 49
6.15 Stacked area plot of the evolution probabilities of different operations in the augmentation bag $\mathcal{T}$ when the ranking is based on a single downstream task (each row in Figure 6.14) 50
6.16 Difference in Average Performance of RU over a fixed augmentation pipeline on SimCLR at various batch sizes. 51
6.17 Average Performance of RU over a fixed augmentation pipeline on SimCLR for different batch sizes. 51
6.18 Difference in Average Performance of RU over a fixed augmentation pipeline on SimCLR at various pre-training epochs. 52
6.19 Average Performance of RU over a fixed augmentation pipeline on SimCLR for different pre-training epochs. 52
6.20 Difference in Average Performance of RU over a fixed augmentation pipeline on SimCLR for different encoders. 53
6.21 Average Performance of RU over a fixed augmentation pipeline on SimCLR for different encoders. 54

A.1 Example images from various datasets. Some images may appear blurry as they have been resized to match the resolution of other images. The original sizes of images from each dataset are provided in Table A.1. . . 69

List of Tables

6.1 Table of Experiments 35
6.2 Top-1 Accuracy with linear evaluation on downstream tasks for our method, ranked based on KL-Divergence and updated using the Top-K update rule ($K=2$). Note that $*$ means *non-statistically significant difference* . . . 41
6.3 Top-1 Accuracy with KKN evaluation on downstream tasks for our method, ranked based on KL-Divergence and updated using the Top-K update rule ($K=2$). 41
6.4 Top-1 Accuracy with linear evaluation on downstream tasks for our method, ranking based on KL-Divergence and update using the Probabilistic update rule ($K=2$). Note that $*$ means *non-statistically significant difference* 43
6.5 Top-1 Accuracy with KNN evaluation on downstream tasks for our method, ranking based on KL-Divergence and update using the Probabilistic update rule ($K=2$). Note that $*$ means *non-statistically significant difference* 43
6.6 Encoder architectures used during pre-training with SimCLR. Each architecture has two properties attached. **#Heads** which represents the number of heads used for Transformer-based architectures and **#Params** which represents the number of parameters in the encoder. 53

A.1 Information about different datasets used in our experiments. 68
A.2 Hyperparameters for Pre-training 69
A.3 Hyperparameters for Linear Evaluation 70

Chapter 1

Introduction

An important aspect of machine learning involves capturing a low-dimensional vector representation space of data to perform various decision-making tasks. The recent surge in the popularity of neural networks, combined with the accessibility of large datasets and substantial computational power have facilitated the use of deep supervised learning for this purpose [Krizhevsky *et al.* 2012; Vaswani *et al.* 2017; Mnih *et al.* 2013]. However, a major hindrance to the broad application of these techniques is the necessity for extensive labeled datasets for each specific task, which entails time-consuming data collection and annotation efforts [Seltzer and Zhang 2009].

Recognizing the challenges of labeling vast amounts of data, researchers have developed techniques that can learn a low-dimensional vector representation space directly from unlabeled data, thus removing the annotation bottleneck [Le *et al.* 2011; Bengio *et al.* 2013]. Some of these techniques go a step further by allowing the model to learn representations that can be re-used for a variety of tasks, rather than needing to be re-learned for each specific task [Erhan *et al.* 2010; Collobert *et al.* 2011; Bengio 2012].

Self-Supervised Learning has emerged as a promising approach for learning this representation space from unlabeled data [Chen *et al.* 2020b; Misra and Maaten 2019; Noroozi and Favaro 2016; Doersch *et al.* 2015]. This technique works by designing *unsupervised tasks* that automatically generate pseudo-labels from the unlabeled data itself. A machine learning model is then trained on these pseudo-labels to make predictions. The design of these unsupervised tasks influences the model's ability to extract meaningful semantic information from the unlabeled data. This information is captured within the model's learned representation space. The key benefit of SSL lies in its ability to transfer this learned knowledge to supervised learning tasks, even when labeled data for those supervised tasks is scarce. Examples of unsupervised tasks used in SSL include predicting image rotation [Gidaris *et al.* 2018], recovering masked-out image regions [He *et al.* 2021], and encouraging similarity in vector representations of image transformations [Chen *et al.* 2020b].

In recent years, there has been a growing emphasis on performing unsupervised tasks that encourage similar vector representations, particularly for solving supervised down-

stream tasks such as classification and object detection [Chen *et al.* 2020b; Caron *et al.* 2020; Oquab *et al.* 2024; Grill *et al.* 2020; Zbontar *et al.* 2021]. These methods are commonly referred to as Contrastive SSL techniques. The fundamental idea is that a robust low-dimensional vector representation space should contain enough information to distinguish between different data points (images in this scenario).

While some supervised techniques can readily achieve this objective when class labels are available [Schroff *et al.* 2015; Sohn 2016; Gutmann and Hyvärinen 2010; Frosst *et al.* 2019], Contrastive SSL lacks this luxury due to the absence of labels. A proposed solution is to learn these representations by encouraging views from the same unlabeled image to have similar vector representations [Chen *et al.* 2020b; Caron *et al.* 2020; Oquab *et al.* 2024; Grill *et al.* 2020; Zbontar *et al.* 2021]. The core concept is rooted in the notion that different views of the same image inherently share identical underlying semantic content.

Therefore, a crucial aspect of these techniques lies in developing an effective mechanism to generate diverse views of the same image. We emphasize this because the information encoded in the representations is influenced by the properties altered in these generated views. For instance, if our view generator changes a property, denoted as *property X* (such as color, orientation, blurriness, etc.), the result will be images with different variations of *property X*. Promoting similarity in the learned representations of these views encourages the representations to ignore *property X* and focus on other properties that aid in similarity prediction. In such cases, we say the learned representations are invariant to *property X*.

Therefore, since we learn these representations to solve a set of supervised downstream tasks, it is crucial to design a view generator that considers these tasks during its view-generating process. Unfortunately, most methods rely on a hand-crafted data augmentation pipeline to create such views [Chen *et al.* 2020b; Caron *et al.* 2020; Oquab *et al.* 2024; Grill *et al.* 2020; Zbontar *et al.* 2021]. These augmentation pipelines typically include a Random Cropping mechanism followed by some color and geometric image transformations. However, those attempting to innovate with novel automatic augmentation methods often fall short due to either a lack of diversity in the generated views [Tian *et al.* 2020] or a computationally expensive approach that introduces a secondary optimization method alongside the SSL process [Reed *et al.* 2021].

To address this problem, we propose a less computationally expensive data augmentation method for contrastive SSL. This approach enables the view generator to consider the downstream tasks of interest during the view-generation process. Specifically, our method employs a rank-update mechanism to periodically *rank* data augmentation operations based on their relevance to different downstream tasks. Subsequently, it *updates* the augmentation pipeline of the unsupervised task based on this ranking. Note that since we use some knowledge of the downstream tasks to perform the ranking, our method might require labels from the tasks in some instances, but we make it clear when this is the case. We demonstrate the effectiveness of our method compared to (1) a variant of the de-facto standard hand-crafted augmentation pipeline and (2) other state-of-the-art augmentation methods.

We believe our method would be highly advantageous in situations where we have a clear understanding of the specific supervised downstream tasks we intend to undertake but lack sufficient data for direct supervised learning training. In such cases, we can conduct contrastive SSL training on unlabeled data while integrating our augmentation method, which considers the intended set of downstream tasks. This ensures that the learned representations are more closely aligned with the targeted downstream tasks of interest.

We make the following contributions:

1. We provide a quantitative and qualitative analysis of the effect of data augmentation on contrastive SSL.
2. We introduce a novel data augmentation framework for contrastive SSL that allows the view generators to consider the downstream tasks of interest during the view-generation process.
3. We demonstrate that our method can learn more effective representations for our chosen set of downstream tasks than those learned by current state-of-the-art augmentation methods such as RandAugment [Cubuk *et al.* 2020] and TrivialAugment [Müller and Hutter 2021],.

This dissertation is structured as follows: Chapter 2 provides a comprehensive overview of the SSL framework, giving more details about its contrastive approaches and their dependency on data augmentation. Chapter 3 discusses various approaches to improve data augmentation, first in supervised learning and then self-supervised learning. Chapter 4 presents our novel data augmentation framework for contrastive SSL, and Chapter 5 details the experimental setup used to evaluate our method. Finally, Chapter 6 presents the results of our experiments, and Chapter 7 concludes our work while providing potential avenues for its expansion in other areas of SSL.

Chapter 2

Background

Before delving into the details of our augmentation framework, this chapter aims to establish an understanding of contrastive SSL and its dependence on data augmentation. In Section 2.1, we provide a comprehensive overview of the SSL framework, drawing parallels with supervised learning. Progressing to Section 2.2, our focus sharpens on the contrastive approaches within the field, offering a detailed account of the various methods employed. Lastly, in Section 2.3, we explore the pivotal role of data augmentation in contrastive SSL, emphasizing its significance within the field.

2.1 Self-Supervised Learning (SSL)

In the domain of deep learning, the primary focus has long been on developing a meaningful representation space[1] from data to perform predictions for various computational tasks [Krizhevsky *et al.* 2012; Mnih *et al.* 2013]. Initially, this was achieved through supervised learning only; however, it became evident that the limited availability of labelled data posed a significant constraint [Krizhevsky *et al.* 2012; He *et al.* 2016]. Moreover, the need to train a new model from scratch for each task emerged as a critical impediment to the widespread applicability of these techniques.

This realization led to the emergence of a novel paradigm in deep learning known as SSL [Doersch *et al.* 2015; Gidaris *et al.* 2018; Caron *et al.* 2018]. SSL involves learning a representation space from unlabeled data through hand-crafted, unsupervised learning tasks. The acquired representation space, whether used in its raw form or fine-tuned further, can then be applied to make predictions on another set of tasks, referred to as *downstream tasks*. In our context, a downstream task refers to a computer vision task such as image classification.

[1] A *Representation Space* is a geometric space where a machine learning model transforms data to generate meaningful representations that capture relevant features of the data.

2.1.1 The SSL Formulation

The terms used in this formulation are adapted from Balestriero *et al.* [2023].

In supervised learning, we wish to minimize the following objective:

$$\min_{\theta} \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}_{sup}}[\mathcal{L}_{sup}(f_\theta(x_i), y_i)], \tag{2.1}$$

where f_θ is a machine learning model with adjustable parameters θ. This minimization is achieved through a supervised objective loss function $\mathcal{L}_{sup}$ applied to a labeled dataset $\mathcal{D}_{sup} = \{(x_i, y_i)\}_{i=1}^{N}$, where each x_i represents a data point and y_i is its corresponding annotation.

What sets SSL apart is its modification of the traditional supervised learning approach, as outlined in Equation 2.1. Specifically, while the labels y_i remain consistent, the input data x_i undergoes transformation by an SSL model f^{SSL}, resulting in a lower-dimensional vector representation $z_i \in \mathbb{R}^d$. It is worth noting that in certain sections of the thesis, z_i will also be referred to as the *latent representation* of x_i. This transformation performed by f^{SSL} occurs within a representation space learned during a process known as *pre-training*, which utilizes unlabeled data $\mathcal{D}_{pre} = \{x_i\}_{i=1}^{N}$. This process can be viewed as a dimensionality reduction technique. Further details about this process are provided in Section 2.1.2.

Consequently, the updated objective becomes:

$$\min_{\theta} \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}_{sup}}[\mathcal{L}_{sup}(f_\theta(f^{SSL}(x_i)), y_i)]. \tag{2.2}$$

Additionally, it is important to note that the architecture of f_θ may need to be updated to accommodate the output from $f^{SSL}(x_i)$. In certain scenarios, the representation space learned by f^{SSL} can undergo further refinement by incorporating its learnable parameters into the minimization objective outlined in Equation 2.2. This process is commonly referred to as *Fine-Tuning*. Alternatively, the representation space learned by f^{SSL} can be leveraged to obtain vector representations of $x \sim \mathcal{D}$, which can serve as features for any statistical model chosen based on the specific task at hand. We delve into the details of both approaches in Section 2.1.2.

2.1.2 SSL in Practice

As mentioned earlier, the typical approach to SSL involves initially learning a representation space with f^{SSL} from unlabeled data. Subsequently, we apply the learned space to a particular downstream task or a set of them. In this section, we delve into the various steps within the SSL pipeline, focusing on computer vision tasks. The process is illustrated in Figure 2.1.

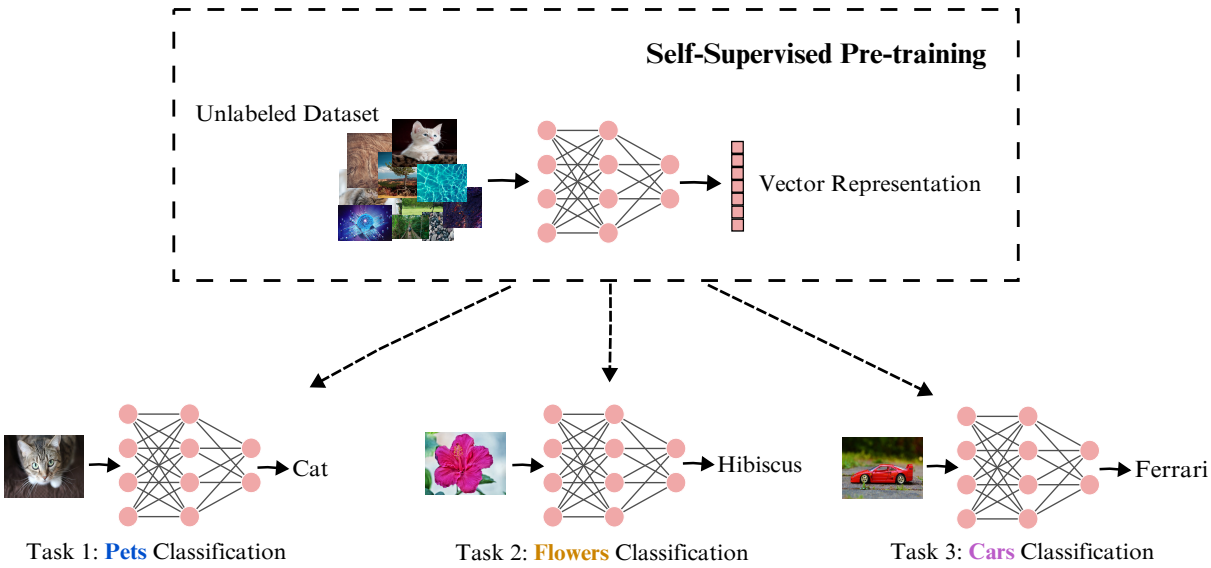


Figure 2.1: The Self-Supervised Learning pipeline. A neural network is trained on an unlabeled dataset through self-supervised pre-training (neural network at the top). Then, the trained network is used to perform different downstream tasks through the updated supervised objective in Equation 2.2 (copies of the network at the bottom).

Pretraining

This is the first step in our SSL pipeline. During this phase, we train an *encoder* f^{SSL} on an unsupervised task using unlabeled data $\mathcal{D}_{pre}$. The objective here is to enable the output of f_{SSL} to capture semantic information about $\mathcal{D}_{pre}$. In other words, we want f_{SSL} to learn a latent representation space for a downstream task set.

In some cases, the output space of f_{SSL} may not align with the input space required for the unsupervised task. For instance, if f_{SSL} maps images to a 128-dimensional space, but the unsupervised task necessitates predictions in a 2-dimensional space (image rotation prediction between angles $[45°, 90°]$), then a *projection head* $g : \mathbb{R}^{128} \rightarrow \mathbb{R}^2$ is added on top of f_{SSL}. The projection head maps the output from f_{SSL} to the desired space.

We aim to optimize both f^{SSL} and g for the following objective:

$$\min_{\theta_1,\theta_2} \mathbb{E}_{x\sim\mathcal{D}_{pre}} \left[\mathcal{L}_{SSL}\left(g_{\theta_2}(f^{SSL}_{\theta_1}(x)), h(x)\right)\right], \tag{2.3}$$

where $\mathcal{L}_{SSL}$ is the loss function for the unsupervised task, h is a function that provides the supervisory signal from the unlabelled data with a process unknown to both f_{SSL} and projection head g. Algorithm 1 provides a pseudocode of the pre-training process, and Section **??** provides a discussion on different techniques used for this process.

Algorithm 1 SSL Pre-Training Process (projector head version)

Load unlabeled dataset $\mathcal{D}_{pre}$
Initialize encoder f^{SSL} and projection head g with parameters θ_1 and θ_2, respectively
Initialize SSL loss function $\mathcal{L}_{SSL}$
while not converged **do**
    Get batch of data X from $\mathcal{D}_{pre}$
    Compute the supervisory signal: $y = h(X)$
    Compute encoder output: $z = f_{SSL}(X)$
    Project the encoder's output: $z' = g(z)$
    Compute gradients over loss: $\nabla_{\theta_1,\theta_2}\mathcal{L}(z', y)$
    Update model parameters: (θ_1, θ_2)
end while
Output: Trained encoder f_{SSL}

Evaluation Protocols

Once the encoder f^{SSL} is trained, it can be applied to the intended downstream tasks using the available labeled data $\mathcal{D}_{sup} = \{(x_i, y_i)\}_{i=1}^{N}$ for each task. Depending on the size of the downstream task data and the computational resources allocated for training, we can choose from three evaluation protocols to assess the performance of the SSL method used to train f^{SSL}. These methods are described as follows:

- **Linear Evaluation**: This method utilizes the encoder as a feature extractor, extracting features that serve as input to any linear statistical model, such as a linear regressor or classifier. It is suitable for cases where computational resources or labeled data are limited for transfer learning with the pre-trained encoder. However, it may not be suitable for tasks that require a more complex decision boundary necessitating a non-linear model.

- **Fine-Tuning**: The second approach involves fine-tuning the encoder for the particular downstream task. This approach is particularly useful when the goal is to adapt the learned space to better suit the specific downstream task without concern for its performance on other tasks. It requires more computational resources than linear evaluation but is more effective for tasks that require a more complex decision boundary. In such cases, the pre-trained encoder acts as a good initialization point for the downstream task of interest.

- **K-Nearest-Neighbors**: This approach closely resembles **Linear Evaluation** in that it also utilizes the encoder as a feature extractor. However, instead of fitting a linear model, we utilize a model based on the K-Nearest Neighbors (KNN) algorithm. This approach is particularly useful when we seek to evaluate the quality of the learned representation space in a non-parametric setting that does not require learning any model parameters.

The objective remains as in Equation 2.2 in both cases. However, for fine-tuning, the optimization process includes the parameters of the encoder f^{SSL}.

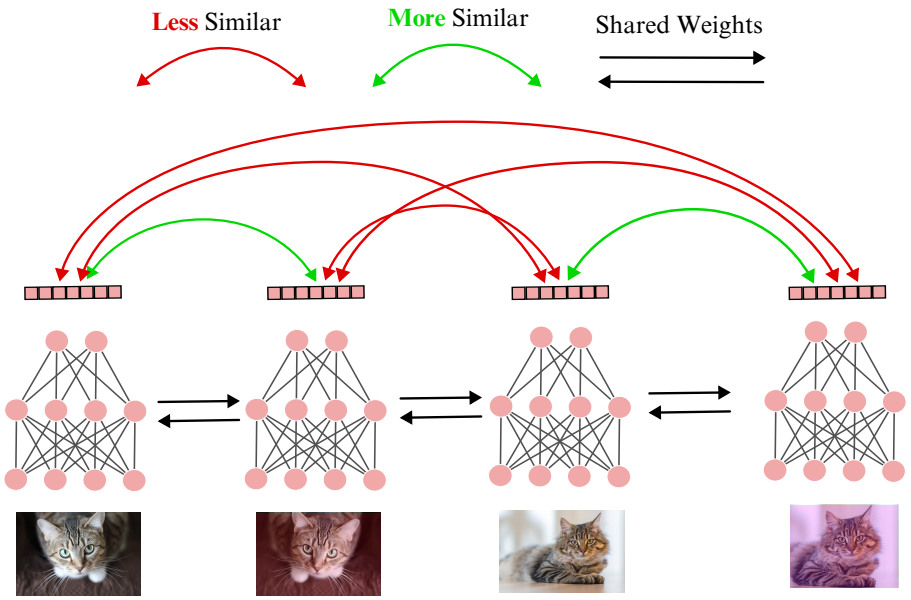


Figure 2.2: A contrastive SSL method where a neural network is trained to bring together the representations of views generated from the same data point while pushing apart the representations of views generated from different data points.

2.2 Contrastive SSL

In recent years, contrastive learning has become increasingly popular in SSL methods during the pre-training process [Chen *et al.* 2020b; Caron *et al.* 2020 2021; Grill *et al.* 2020]. The core idea of contrastive learning is that optimal data representations should contain enough information to distinguish between different data points. It aims to create representations that strip away irrelevant details, focusing on essential discriminative features that foster a more robust and generalized understanding of the underlying data structure, thereby enhancing the model's ability to recognize meaningful patterns across various downstream tasks.

The fundamental principle of contrastive SSL is to create effective data representations that can differentiate between various data points [Jaiswal *et al.* 2020]. This is achieved using a contrastive loss function, which encourages the representations of similar data points to be closer while pushing apart those of dissimilar data points. Typically, similar data points are generated by passing the same data point through a randomized data augmentation pipeline. The representations of views from the same data point are called *positive samples*, while those from different data points are called *negative samples*. Following our contrastive objective function, positive samples are attracted, and negative samples are optionally repelled. Notably, some contrastive SSL methods solely utilize positive samples and do not include negatives in their contrastive objective [Caron *et al.* 2020 2021; Grill *et al.* 2020; Zbontar *et al.* 2021]. Both cases are elaborated upon in Section 2.2.1 and Section 2.2.2.

The contrastive version of the SSL pre-training process, as depicted in Algorithm 1, is presented in Algorithm 2. The main modifications from the original algorithm include generating multiple views from a batch of data points using an augmentation pipeline

sampled from τ (in line 7). In addition, we use a contrastive loss function on the representations z' obtained from those views (in lines 10 and 13).

Algorithm 2 Contrastive SSL Pre-Training Process. The green and red highlights indicate the lines that were added and removed from Algorithm 1, respectively.

1: Load unlabeled dataset $\mathcal{D}_{pre}$
2: Initialize encoder f^{SSL} and projection head g with parameters θ_1 and θ_2, respectively
3: $-$ Initialize SSL loss function $\mathcal{L}_{\text{SSL}}$
4: $+$ Initialize SSL contrastive loss function $\mathcal{L}_{\text{contrastive SSL}}$
5: **while** not converged **do**
6: Get batch of data X from $\mathcal{D}_{pre}$
7: $+$ Generate multiple views with $t \sim \tau$ from X: $\{X_1, X_2, \ldots, X_n\}$
8: $-$ Compute the supervisory signal: $y = h(X)$
9: $-$ Compute encoder output: $z = f_{SSL}(X)$
10: $+$ Compute encoder output: $z = f_{SSL}(X_1, X_2, \ldots, X_n)$
11: Project the encoder output: $z' = g(z)$
12: $-$ Compute gradients over loss: $\nabla_{\theta_1,\theta_2}\mathcal{L}_{\text{SSL}}(z', y)$
13: $+$ Compute gradients over loss: $\nabla_{\theta_1,\theta_2}\mathcal{L}_{\text{contrastive SSL}}(z')$
14: Update model parameters: (θ_1, θ_2)
15: **end while**
16: **Output:** Trained encoder f_{SSL}

2.2.1 With Negatives

Methods in this category originate from the early days of contrastive learning [Chopra *et al.* 2005]. They take advantage of the knowledge acquired in deep metric learning [Schroff *et al.* 2015; Sohn 2016; Gutmann and Hyvärinen 2010; Frosst *et al.* 2019] and apply it in an unsupervised context. In deep metric learning, the objective is to contrast between different data points. Therefore, the objective function encourages the representation of samples from the same class to be pushed together while pushing away samples from different classes.

However, in a self-supervised setting, we do not have any labels. This creates the necessity to find a way to contrast between the samples we have. The most popular solution to this problem is to generate views from the data points themselves and pull together the representations of the views generated from the same data point while pushing away the representations of the views generated from different data points.

In this context, views from the same data points are called positive samples, while those coming from different data points are called negative samples. Therefore, we aim to attract positive samples and repel negative samples. Contrastive SSL techniques using negative samples have been shown to learn embedding spaces where semantically similar data points are close to each other [van den Oord *et al.* 2018; Chuang *et al.* 2020].

2.2.2 Without Negatives

The main disadvantage of methods that use negative samples is their dependence on the batch size. This dependence comes from the current training batch being the only source of negative samples. Therefore, the frequency of model parameters' updates depends on the batch size. A solution to this problem could be to decouple the dependence on the number of updates from the batch size by storing negatives in a queue or other storage facility utterly separate from the current training batch of data [Wu *et al.* 2018; He *et al.* 2020; Chen *et al.* 2020c].

Other methods propose to learn representations without negatives [Caron *et al.* 2021; Chen and He 2021; Grill *et al.* 2020]. The main idea behind these methods is to learn representations invariant to certain transformations of the same data point without forcefully pushing away the representations of other data points. However, a weakness of such methods is the tendency of the model to collapse to a trivial solution where the representations of all the data points are the same, as this is the easiest way to reach the global optimum [Jing *et al.* 2022].

Some methods address the potential collapse of representations to a trivial solution through the use of a stop-gradient mechanism [Caron *et al.* 2021; Chen and He 2021; Grill *et al.* 2020; Caron *et al.* 2020]. This technique involves preventing the back-propagation of gradients through certain operations or components of the neural network during training, serving as a form of regularization to encourage more diverse and meaningful representations. Alternatively, others opt for a clustering-based loss to guide the learning process [Caron *et al.* 2020]. In such an approach, we do not pull the representations of different views from the same image towards each other. Instead, we pull them towards a single cluster vector representation, which serves as a reference point that captures the commonalities between different views of the same image [Caron *et al.* 2020].

2.3 Data-Augmentation in Contrastive SSL

While extensive research is dedicated to data augmentation in supervised learning [Cubuk *et al.* 2019; Lim *et al.* 2019; Hataya *et al.* 2019; Li *et al.* 2020; Gudovskiy *et al.* 2021; Zheng *et al.* 2022; Mehta *et al.* 2022; Müller and Hutter 2021; Cubuk *et al.* 2020], there remains a noticeable gap in the literature regarding methods to intelligently perform data augmentation within contrastive SSL, despite its recognized significance [Tian *et al.* 2020; Wagner *et al.* 2022; Chen *et al.* 2020b].

In contrastive SSL, the objective is to learn optimal data representations that can effectively distinguish between different data points. This is primarily achieved through a contrastive objective, which aims to maximize the similarity between the representations of semantically similar data points. However, a significant challenge in this SSL setting is the lack of a supervisory signal to identify these similar data points.

A well-known technique to address this challenge is to generate multiple views of the same image and encourage their representations to be similar. Formally, given an input sample x from the distribution X and a distribution of augmentation operations T, we

sample two independent transformations t_1 and t_2 from T. When these transformations are applied to x, they produce x_1 and x_2, respectively. This approach is used in methods that rely solely on positive examples.

In methods that use both positive and negative samples, a second image y is involved. Applying the same transformations to y results in y_1 and y_2. For the positive pairs, the objective is to increase the similarity of the vector representations of x_1 and x_2. For the negative pairs, the objective includes decreasing the similarity between the vector representations of the following pairs: (x_1, y_1), (x_1, y_2), (x_2, y_1), and (x_2, y_2). This process is illustrated in Figure 2.2.

Data augmentation operations t_1, t_2 in contrastive SSL introduces variations in the views generated from the same data point. Encouraging the representations of these views to be similar is akin to promoting learned representations that overlook the introduced variance of information caused by the data augmentation operations, resulting in the trained model being ill-equipped to perform tasks that necessitate such information in the vector representations of their data points [Tian *et al.* 2020]. However, it accelerates learning on tasks that do not require this information, as they do not need to re-learn these invariances.

Therefore, in contrastive SSL, the selection of data augmentation plays a pivotal role in determining the invariance learned by the model. For instance, an operation that distorts the color of an image will prompt the model to acquire representations invariant to color. Nevertheless, this may prove unhelpful for a downstream task that requires color information, as all the essential details for making predictions have been removed from the representations during pre-training.

2.4 Summary

This chapter presented SSL, setting the stage to explore data augmentation in contrastive SSL. The chapter emphasized the importance of obtaining meaningful image views for effective learning within contrastive SSL. A critical aspect highlighted was the reliance on the appropriate augmentation pipeline to generate these views, recognizing the impact of data augmentation on the invariance learned by the model. This insight provides an appropriate transition to Chapter 3, where we describe prior solutions to the current limitations of data augmentation in contrastive SSL.

Chapter 3

Related Work

Data Augmentation plays a crucial role in contrastive SSL by shaping the learned representation space [Ericsson *et al.* 2022]. However, it serves as a means to an end, as the ultimate goal is to learn representations that generalize well to various downstream tasks. Many researchers have recognized data augmentation as an inherent limiting factor in achieving this goal and have consequently proposed new techniques to reduce reliance on it [Assran *et al.* 2023; He *et al.* 2021]. These methods are discussed in detail in Section 3.1. Others view this limitation as an opportunity to introduce novel ideas to enhance data augmentation in contrastive SSL [Reed *et al.* 2021; Wagner *et al.* 2022]. These methods are also described in Section 3.2. Although our work aligns with the latter group, our aim is to provide a comprehensive overview of the research landscape on this topic. Figure 3.1 illustrates the main design decisions in our work, which are contextualized by the literature in the sections that follow.

3.1 Ignoring Data Augmentation

The effectiveness of contrastive SSL methods hinges significantly on their capacity to learn data invariances from the data augmentation pipeline, as highlighted by recent studies [Ericsson *et al.* 2022; Wagner *et al.* 2022]. However, the conventional approach to selecting an appropriate augmentation pipeline often involves experimenting with various combinations of data augmentation operations, choosing the one that yield the best performance [Chen *et al.* 2020b; Ericsson *et al.* 2022]. In Section 3.2, we explore methods proposed to streamline this computational burden.

Some approaches completely change direction to address this challenge. Instead of persistently seeking the optimal augmentation pipeline, these methods diverge from the contrastive approach entirely. Meanwhile, others, while still embracing the contrastive approach, introduce equivariance constraints in addition to the existing invariance constraints in the learned representations through a modified contrastive learning objective. We thoroughly examine both approaches in Section 3.1.1 and Section 3.1.2.

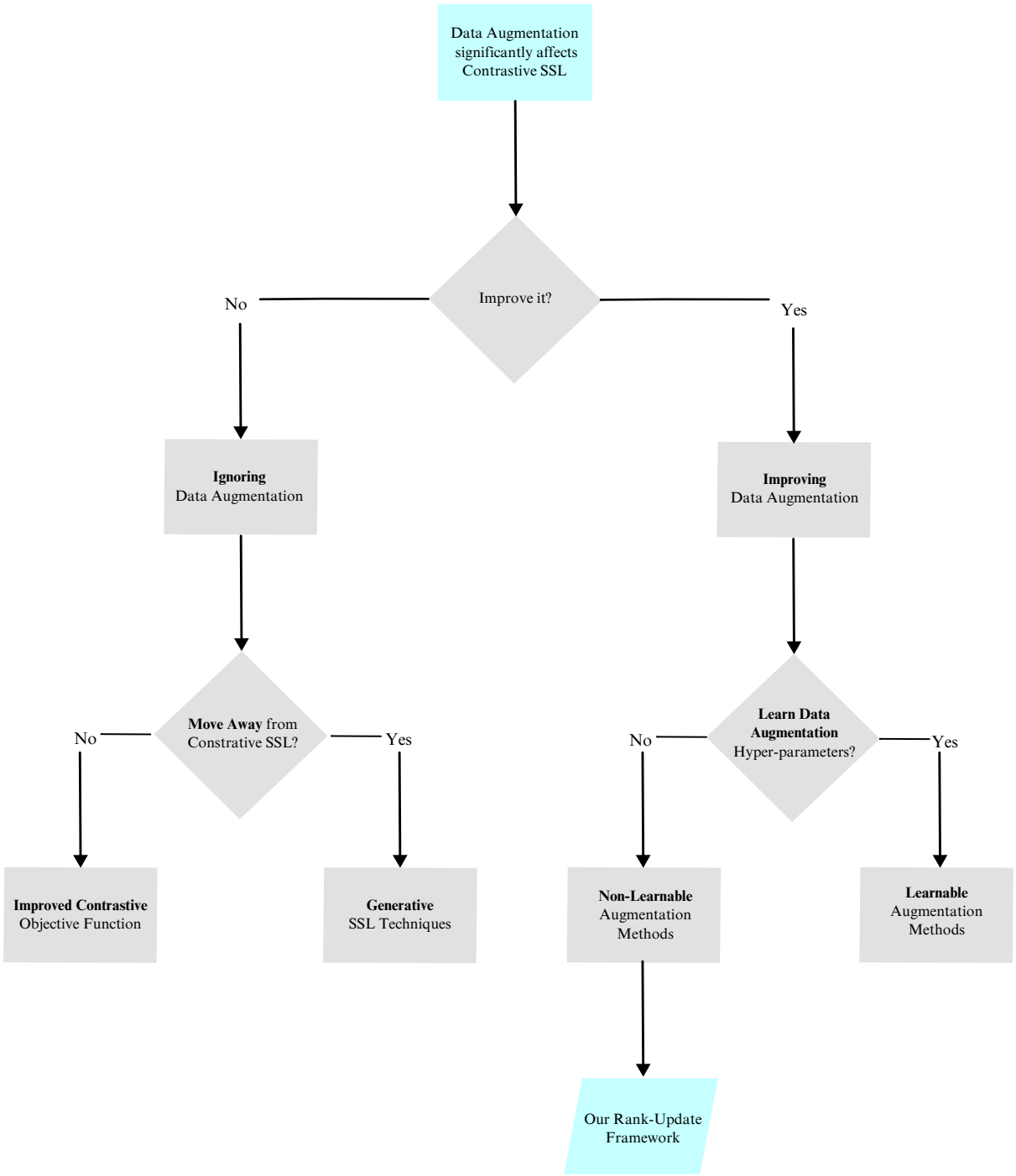


Figure 3.1: Roadmap of the related work section leading to our proposed approach

3.1.1 Generative SSL

Before the recent upsurge in the prevalence of contrastive SSL methods, a foundational paradigm within SSL entailed reconstructing input data from latent vectors [Pathak *et al.* 2016; Mikolov *et al.* 2013]. These methods utilized auto-encoding architectures, passing input data through an encoder to a latent space and reconstructing it from the latent space through a decoder [Baldi 2011; Kingma and Welling 2013]. Post-training, the decoder is discarded, and the encoder is repurposed for diverse tasks. The underlying presumption is that training a network to reconstruct input data from a lower-dimensional latent space allows the network to encapsulate the quintessential features of the data within this space, facilitating subsequent task performance. These techniques have demonstrated noteworthy efficacy, particularly in generative tasks such as image generation [Razavi *et al.* 2019; Sidheekh *et al.* 2022].

Nevertheless, some may question the oversimplified nature of these techniques. In essence, an encoder-decoder architecture employed in such tasks is equivalent to learning a mapping function f (encoder) and its inverse f^{-1} (decoder). Consequently, this architecture lacks robustness to different perturbations of the input [Vincent *et al.* 2008]. A logical extension to enhance such methods would be to heighten the challenge of reconstructing the original input, perhaps by introducing noise in the input space of the neural network [Vincent *et al.* 2008]. This approach has proven highly effective in learning representations for Natural Language Processing (NLP) tasks [Sanh *et al.* 2019; Libovický *et al.* 2020]. Such methods in NLP are categorized under the umbrella of **masked language modeling** techniques.

While these techniques have demonstrated tremendous success in NLP, it is crucial to recognize their longstanding presence in computer vision. Notably, Vincent *et al.* [2010] proposed one of the earliest denoising autoencoder methods within computer vision, focusing on learning high-level representations from corrupted images. What sets apart generative methods in NLP from this early work is the introduction of the Transformer architecture [Vaswani *et al.* 2017]. The Transformer's distinctive lack of a strong inductive bias enabled it to learn neural representations from large-scale datasets [Sanh *et al.* 2019; Libovický *et al.* 2020; Devlin *et al.* 2019; Conneau *et al.* 2019].

To emulate this success in computer vision, Chen *et al.* [2020a]; Dosovitskiy *et al.* [2020] propose methods employing a transformer architecture for a pixel prediction task using masked input. However, a notable drawback of such approaches lies in the reconstruction loss, which occurs at the pixel level, consequently amplifying the computational demands of these methods. The challenge arises from the model needing to learn the spatial relationships between pixels in the image, an operation that scales quadratically with the number of pixels [Dosovitskiy *et al.* 2020]. While this may pose a minor concern for small images comprising only a few pixels, it becomes a pronounced issue for large images containing millions of pixels.

A natural solution emerges when considering how this problem is addressed in NLP. While most language modeling tasks operate at the word level, it is widely acknowledged that words are not the most atomic unit of a language. In fact, characters hold this distinction, and although character-level tasks can be performed in NLP [Zhang *et*

al. 2015; Seo *et al.* 2016; Group 2017], it is often preferable to perform NLP tasks at a word or sub-word level due to the semantic richness associated with words [Mikolov *et al.* 2013]. Similarly, in computer vision, pixels are the most atomic units of an image. However, it is considered more advantageous to regard patches of pixels around different regions of an image as a more semantically rich unit. This concept serves as the foundation for various methodologies in which masked patches of pixels are encoded into latent vectors and subsequently reconstructed from these vectors [Bao *et al.* 2021]. Notably, in certain scenarios, emphasis is placed on reconstructing the latent vectors of the original patches [Zhou *et al.* 2022; Assran *et al.* 2023].

These generative techniques circumvent the need for data augmentation through generative modeling. However, this approach may inadvertently encourage the model to encode more information than required in its latent space, particularly for tasks such as image classification. Consequently, methods that still aim to constrain the information encoded in the latent space often continue to adhere to the contrastive paradigm.

3.1.2 Improved Objective Function

As said earlier, while generative techniques are a promising alternative to contrastive SSL methods, they occasionally encode more information than necessary in the latent space for discriminative tasks like classification and object detection [Liu *et al.* 2021]. In most cases, contrastive learning methods are more effective for such tasks [Chen *et al.* 2020b; He *et al.* 2020; Wu *et al.* 2018]. Consequently, a viable avenue to enhance contrastive SSL methods, notwithstanding their dual reliance on data augmentation, is to refine other components within the contrastive SSL pipeline beyond the augmentation process, mainly focusing on the contrastive objective [Chen *et al.* 2020b; He *et al.* 2020; Addepalli *et al.* 2022; Dangovski *et al.* 2022; Wu *et al.* 2018; Torpey and Klein 2021].

One effective strategy is introducing an additional objective alongside the contrastive objective, encouraging the model to acquire supplementary information beyond what is attainable from the original contrastive objective alone. These methods are particularly advantageous as they can seamlessly integrate into any contrastive SSL approach. For instance, a common approach appends a homography prediction task to the contrastive objective [Torpey and Klein 2021; Wang *et al.* 2021; Addepalli *et al.* 2022], thereby ensuring some level of equivariance in the learned representations. This addition becomes pertinent as representation learning may not solely require invariance but also necessitate equivariance [Dangovski *et al.* 2022]. Similarly, another approach involves adding a rotation prediction task to the original contrastive objective with the hope that promoting equivariance rather than strict invariance to image orientation may prove beneficial for contrastive SSL [Ni *et al.* 2022].

Conversely, specific investigations posit a fundamental flaw in the prevailing contrastive objectives concerning the augmentation pipeline. Some scholars advocate for modifying these objectives to address this issue. Miyai *et al.* [2022] argues that while it makes sense to encourage similarity between the representations of different views of the same image, there are cases where rotated views of specific images should have dissim-

ilar latent representations. Consequently, an effective enhancement to the contrastive loss function involves identifying and adjusting to such cases accordingly. Alternatively, some scrutinize the original contrastive objective function, recognizing a seemingly stringent assumption that all views of the same image should be inherently similar. As a result, efforts are made to relax this constraint to accommodate situations where stronger augmentations might cause a more significant semantic shift between views of the same images [Wang *et al.* 2021; Wang and Qi 2021]. A more radical approach completely departs from the conventional contrastive objective function. For instance, beyond merely enforcing the invariance of learned representations like in the original SSL contrastive function, one could advocate for implementing equivariance through a novel objective function [Bardes *et al.* 2021].

3.2 Improving Data Augmentation

As previously mentioned, data augmentation is pivotal in shaping the learned representation space [Ericsson *et al.* 2022]. However, the brute-force approach to selecting an appropriate augmentation pipeline is often deemed inefficient, and this has led to the rise of methods that aim at streamlining this process, both in the self-supervised setting [Reed *et al.* 2021; Wagner *et al.* 2022] and more prominently in the supervised setting [Cubuk *et al.* 2019; Lim *et al.* 2019; Hataya *et al.* 2019; Li *et al.* 2020; Gudovskiy *et al.* 2021; Zheng *et al.* 2022]. On the other hand, some works lean towards heuristic-based approaches to improve data augmentation [Mehta *et al.* 2022; Müller and Hutter 2021; Cubuk *et al.* 2020]. Unfortunately, the topic of data augmentation in visual representation learning is yet to have a defined formalism to guide the development of such methods. Nevertheless, we do our best to segment different approaches proposed to solve this problem into two categories: *Learnable Methods*, which usually construct augmentation pipelines based on some learned parameters optimized through an objective function, and *Non-Learnable Methods*, whose method for creating the augmentation pipeline does not depend on any learnable parameter.

3.2.1 Learnable Augmentations

From the perspective of data augmentation as a hyperparameter, the problem of learning the optimal augmentation strategy can be viewed as a hyperparameter optimization task. A seminal contribution to this area of research stems from the domain of supervised learning, where Cubuk *et al.* [2019] tackled this problem using tools from deep reinforcement learning (RL). An RL agent creates an optimal augmentation pipeline through its action space. The environment in which this agent operates is a supervised classification task. Thus, the objective of this agent is to discover the optimal augmentation pipeline that maximizes performance on that classification task.

Unfortunately, such methods are often costly as they require a bi-level optimization strategy (training the agent and a model on the task), which, coupled with the sample-inefficient nature of deep RL algorithms makes this process very expensive [Cubuk *et al.* 2019]. On the other hand, when examining data augmentation and the parameters

needed to be learned to construct the augmentation pipeline, we notice that fewer parameters need to be optimized compared to conventional deep learning tasks [Cubuk *et al.* 2019]. This observation highlights the lightweight nature of the optimzation process and questions the necessity of deep RL techniques for such problems. Some works have chosen other optimization methods, relying on Bayesian optimization [Lim *et al.* 2019]. In addition to that, we can make the learning of the augmentation parameters fully differentiable where we directly learn the augmentation parameters concurrently with the model parameters by using back-propagation from the objective function [Hataya *et al.* 2019; Li *et al.* 2020; Hataya *et al.* 2022].

Despite the surge of such methods in supervised learning, only some works still propose using such techniques in SSL. This is mainly due to the complexity of optimizing both the augmentation pipeline and the significant SSL pipeline. In addition, we do not just want to optimize both pipelines for one task but for as many supervised tasks as possible. However, the specific downstream tasks for which our pre-trained models are intended might be unknown. A solution is to design some unsupervised evaluation that correlates with the performance on the intended downstream tasks without requiring their labels [Garrido *et al.* 2022; Agrawal *et al.* 2022]. A popular one revolves around learning the augmentation pipeline using a simple rotation prediction task on the unlabeled images [Reed *et al.* 2021].

On the other hand, we still have the problem of this bi-level optimization strategy, which remains an expensive procedure in its own right, requiring multiple iterations of the pre-training and downstream task evaluation process to find the optimal augmentation pipeline. Therefore, some works propose gradient-based methods that directly optimize the augmentation pipeline objective concurrently with the pre-training objective [Tian *et al.* 2020]. Such works provide an optimized augmentation module using an adversarial objective maximizing the contrastive loss [Tian *et al.* 2020]. Unfortunately, few works that follow this direction exist due to the limitations of the types of operations you can optimize if you want some gradients to flow through these operations (color augmentations through 1x1 convolutional neural networks (CNN) filters). However, there have also been attempts to bypass this restriction by using a differentiable relaxation of the augmentation operation to permit gradients to flow through them but only in the supervised setting [Hataya *et al.* 2019; Li *et al.* 2020; Hataya *et al.* 2022].

3.2.2 Non-Learnable Augmentations

Another class of augmentation methods is those that do not optimize any parameters for the augmentation pipeline but rely on some heuristics to construct the augmentation pipeline. Such methods are particularly appealing as they are often very cheap to implement and can be applied to any contrastive SSL method. These methods can be differentiated based on the heuristic they use to construct the augmentation pipeline.

The most common heuristic is one based on a fixed augmentation pipeline. This is the standard in most supervised and SSL methods [Chen *et al.* 2020b; Caron *et al.* 2020; Zbontar *et al.* 2021; Oquab *et al.* 2024; Caron *et al.* 2021]. Such methods are

often based on the notion that the practitioner has some domain-specific knowledge about the data and the task to be performed. Therefore, they can design an augmentation pipeline suitable for the task. This is the case for the standard augmentation pipeline in SSL based on images similar to the popular image datasets such as ImageNet [Russakovsky *et al.* 2015] or JTF-300M [Sun *et al.* 2017]. While this is a very simple heuristic, it is often very effective in practice but needs to be more generalizable to other tasks [Grill *et al.* 2020; Wagner *et al.* 2022].

Some works propose using a more generalizable heuristic to construct the augmentation pipeline. The idea involves a random sampling process of different configurations of an augmentation pipeline to build the pipeline [Grill *et al.* 2020]. Randomizing this process ensures a particular exploration of the input space by the learning algorithm in a way that is not limited by the augmentation pipeline. Such methods often revolve around assigning a probability distribution over the different operations in the augmentation pipeline and sampling from this distribution to construct the augmentation pipeline [Cubuk *et al.* 2020; Müller and Hutter 2021; Wagner *et al.* 2022]. This is the case for the work of Cubuk *et al.* [2020], where a categorical distribution is defined over the operations and the size of the pipeline. As a result, the augmentation's type and strength are varied.

Finally, we do remark that while such works ensure a simple augmentation strategy while remaining challenging enough, the learnable nature of the methods in Section 3.2.1 makes them more appealing because of the ability to adapt to a particular task. As a result, we believe the future of data augmentation in contrastive SSL lies in the direction of the ability to build simple augmentation pipelines like the ones in Section 3.2.2 but parameterized to adapt to the particular downstream task at hand, offering generality to different tasks in the case of contrastive SSL.

3.3 Summary

Considering the significant role of data augmentation in contrastive SSL, we identified two potential research directions: one aims to eliminate dependency on data augmentation, while the other seeks to improve the augmentation process. The first direction has two subgroups: enhancing contrastive objective functions and generative SSL methods. Enhancing contrastive objective functions involves either adding additional objectives or modifying the original to reduce the unwanted effects of data augmentation. Generative SSL methods replace the contrastive objective with a generative one, aimed at reconstructing the original inputs.

The second research direction also has two subgroups: learnable and non-learnable augmentations. Learnable methods treat data augmentation as a hyperparameter optimization problem, using techniques like deep reinforcement learning and Bayesian optimization to learn the parameters for the augmentation pipeline. Non-learnable methods use heuristics to build these pipelines. We highlighted the strengths and limitations of each approach and emphasized the potential of combining the simplicity of non-learnable strategies with the adaptability of learnable methods. In the next chapter, we provide more details on how we plan to achieve this.

Chapter 4

Methodology

In Chapter 2, we presented essential information necessary to understand our work in data augmentation and its implications in contrastive SSL. Notably, we recognized the need for a more intelligent data augmentation approach in contrastive SSL, as discussed in Chapter 3. This chapter first provides a motivation for this work. Then, we introduce the framework, explaining its different parts and how they work together to produce the desired augmentation mechanism for contrastive SSL.

4.1 Main Motivation

Our research is based on the premise that the effectiveness of various augmentation operations in contrastive SSL is closely linked to the characteristics of the downstream tasks for which the representations are intended. We support this proposition with empirical evidence presented in Section 6.1. The augmentation operations applied during pre-training play a crucial role in shaping the learned representation space, introducing specific invariances to SSL models [Bordes *et al.* 2023; Tian *et al.* 2020; Chen *et al.* 2020b]. This significantly influences how well the representation space aligns with the requirements of the different intended downstream tasks. This phenomenon is discussed in detailed in Section 2.3.

Based on this premise, we propose a novel framework that periodically ranks augmentation operations and then utilizes this ranking to construct an augmentation pipeline during pre-training. The aim is for this augmentation pipeline to encourage the SSL model to learn the appropriate invariances in the representation space for the different downstream tasks of interest.

4.2 Modifying Data Augmentation in the Contrastive SSL

In this section, we provide an overview of how we integrate our data augmentation method into the contrastive SSL pre-training phase. Subsequently, we delve into the details of this method in Section 4.3.

In its original form, the pre-training phase primarily involves iteratively sampling a batch of images from an unlabeled dataset, applying the augmentation pipeline to each image in the batch, and passing the augmented images through the objective function of the contrastive SSL algorithm. This process is repeated until a predetermined criterion is met, typically reaching a fixed number of epochs. A more detailed description of this process can be found in Section 2.2.

The slight modification we introduce to this process is the periodic update of the augmentation pipeline used during pre-training. This update occurs every E epochs, where E is the interval at which the augmentation pipeline is updated. The update process involves the ranking of a set of augmentation operations, as explained in Section 4.3.1, and the construction of an augmentation pipeline based on the previously obtained ranking, as explained in Section 4.3.2. The new augmentation pipeline is then used for the subsequent E epochs of the pre-training phase until we reach the next update interval. This process is repeated until the pre-training phase is complete. The interval update parameter E allows us to control the frequency of the ranking and the pace at which the contrastive SSL model adapts to the periodic pipeline updates. A pseudocode of how this method works in contrastive SSL is depicted in Algorithm 3.

Algorithm 3 Contrastive SSL Pre-Training Process (see Algorithm 2) with the Rank-Update Framework

1: Load unlabeled dataset $\mathcal{D}_{pre}$
2: Initialize encoder f^{SSL} and projection head g with parameters θ_1 and θ_2, respectively
3: Initialize SSL contrastive loss function $\mathcal{L}_{\text{contrastive SSL}}$
4: **while** not converged **do**
5: Get batch of data X from $\mathcal{D}_{pre}$
6: **if** rank update interval reached **then**
7: + Load the different downstream tasks dataset $\mathcal{D}_{task}$
8: + Load the set of operations from the augmentation bag $\mathcal{T}$
9: + τ = **rank_update_augmentation_operations**($\mathcal{T}$, f^{SSL}, $\mathcal{D}_{\text{task}}$)
10: **end if**
11: Generate multiple views with $t \sim \tau$ from X: $\{X_1, X_2, \ldots, X_n\}$
12: Compute encoder output: $z = f_{SSL}(X_1, X_2, \ldots, X_n)$
13: Project the encoder output: $z' = g(z)$
14: Compute gradients over loss: $\nabla_{\theta_1,\theta_2}\mathcal{L}_{\text{contrastive SSL}}(z')$
15: Update model parameters: (θ_1, θ_2)
16: **end while**
17: **Output:** Trained encoder f_{SSL}

4.3 Rank-Update Data Augmentation Framework

Given that we have described how our rank-update method fits into the contrastive SSL pre-training phase, we now delve into the details of it. This section explains how τ (the augmentation pipeline) is returned in line 9 of Algorithm 3.

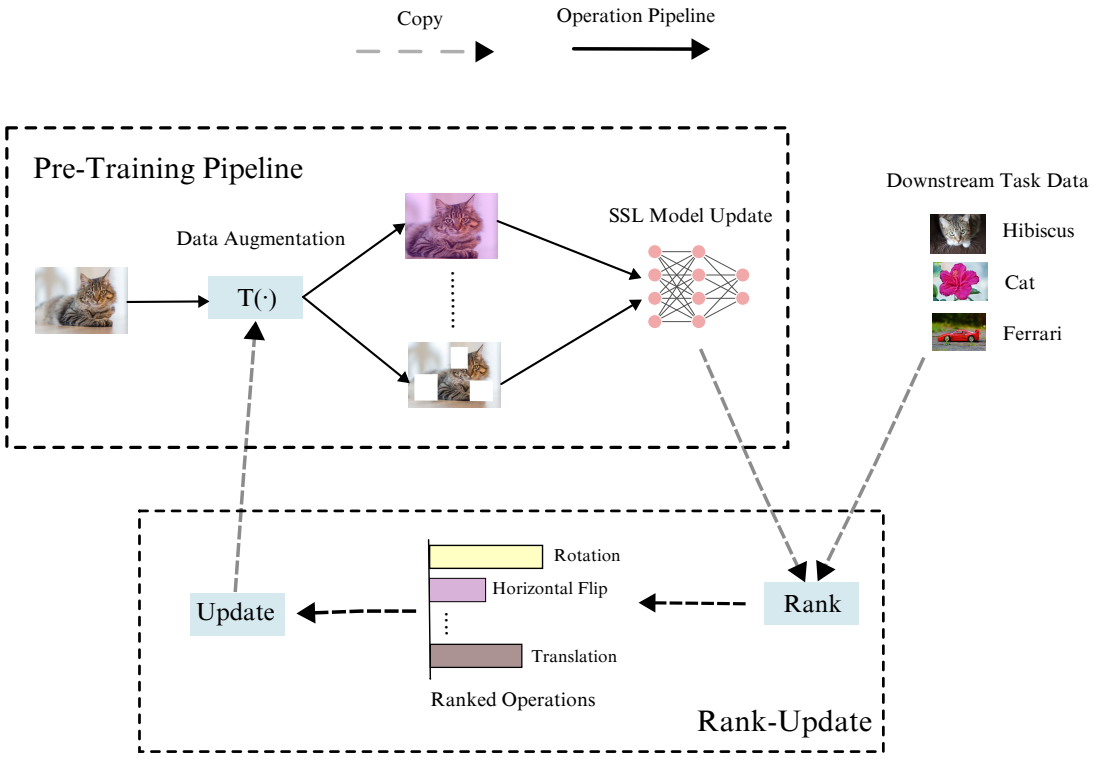


Figure 4.1: Our Rank-Update Framework in the Contrastive SSL pre-training pipeline.

Finding the appropriate augmentation pipeline to enhance model performance can be viewed as a form of hyperparameter optimization [Cubuk *et al.* 2019]. In the context of deep learning, this optimization approach is distinctively bi-level: an inner loop oversees the training of the deep neural network, while an outer loop is dedicated to optimizing the augmentation pipeline's hyperparameters to enhance the inner loop's performance. Notably, this procedure imposes significant computational demands, especially in the context of SSL, where the inner optimization loop requires pre-training and downstream task evaluation (see Section 2.1).

Building upon this insight, we present a novel framework that ranks augmentation operations and uses this ranking to construct the augmentation pipeline during pre-training periodically. It is essential to note that our approach deviates from traditional bi-level hyperparameter optimization processes. Instead of distinct optimization loops for building the augmentation pipeline and training the contrastive SSL algorithm, we concurrently optimize both the augmentation pipeline and the contrastive SSL algorithm, thereby alleviating the high computational demands associated with traditional bi-level optimization.

However, there are potential drawbacks to this approach. Unlike traditional SSL algorithms, where the pre-training phase operates independently of the downstream task evaluation phase, our method requires considering the downstream tasks of interest during the pre-training phase. Furthermore, our approach involves simultaneously optimizing two interdependent objectives: the augmentation pipeline and the pre-training model parameters. The update of the augmentation pipeline depends on the currently

trained SSL model, while this same SSL model is trained with the images generated from the updated augmentation pipeline.

To address these concerns, we incorporate knowledge of the downstream tasks solely in the ranking process (as discussed in Section 4.3.2), while keeping the contrastive pre-training method unchanged. Through this, our rank-update method solely focuses on utilizing the current state of the SSL model, the set of downstream tasks $\mathcal{D}_{\text{task}}$, and the augmentation bag $\mathcal{T}$ to build an augmentation pipeline. Our framework is visually depicted in Figure 4.1.

4.3.1 Ranking Mechanism

The ranking mechanism serves as a preliminary step to our update process. Its purpose is to assist in identifying which operations are most relevant for the intended downstream task, and therefore, which ones should be included in the augmentation pipeline. It evaluates each operation $t \in \mathcal{T}$ in the augmentation bag using a scoring function $f_{\text{rank}}(\cdot)$. This function considers both a downstream task $d_{\text{task}} \in \mathcal{D}_{\text{task}}$ and the currently pre-trained encoder f_{enc}. By applying this scoring function f_{rank} pairwise to every combination of (t, d_{task}), we obtains the average relevance for each operation $t \in \mathcal{T}$ across different downstream tasks of interest.

Algorithm 4 Ranking Augmentation Operations

```
Load set of downstream tasks D_task
Get currently trained encoder f_enc and augmentation bag T
Create 2D array of scores of size |D_task| × |T|
for task in D_task do
    for t in T do
        score ← f_rank(t, f_enc, task)
        scores[task, t] ← score
    end for
end for
scores ← average_over_tasks(scores)
probabilities ← softmax(scores)
Output: probabilities over t ∈ T
```

Please note that the scoring function f_{rank} is an arbitrary function designed to capture the relationship between augmentation operations and downstream tasks. This function is not learned and can be designed based on any assumption regarding the relevance of an augmentation operation to a specific downstream task. In Section 4.3.1, we propose a simple scoring function based on KL-Divergence. However, there is little understanding of how the nature of a downstream task affects the relevance of an augmentation operation [Ericsson *et al.* 2022; Bordes *et al.* 2023]. Consequently, most processes for designing a suitable augmentation pipeline either rely on domain-specific knowledge or heuristic-based methods. If such knowledge could be defined mathematically, then designing a suitable ranking function would be more straightforward. Therefore, we consider the design of an optimal f_{rank} that best captures the relevance

of augmentation operations to particular downstream tasks as an open challenge for anyone willing to extend this work.

The outcome of this ranking process is a softmax-normalized set of scores, one for each operation in $\mathcal{T}$. Operations with the highest scores are expected to be more prevalent in the constructed augmentation pipeline. To accomplish this, we utilize these scores to construct the augmentation pipeline in the subsequent step of our rank-update process through the *Update Mechanism* (see Section 4.3.2). A pseudocode of the ranking mechanism is provided in Algorithm 4.

f_{rank} based on KL-Divergence of Similarities

It is important to note that we can use any function that accepts as input, a pre-trained model, a downstream task (consisting of images and optionally, corresponding annotations), and an augmentation operation, and returns a non-negative real number. However, we propose a method based on KL-Divergence due to following reason.

Consider a scenario where we have a set of three images denoted as $X = \{x_1, x_2, x_3\}$. Additionally, we possess two augmentation operations, t_1 and t_2. When we apply each operation to every images in the set, we obtain two additional sets of images represented as X^{t_1} and X^{t_2}. Specifically, $X^{t_i} = \{x_1^{t_i}, x_2^{t_i}, x_3^{t_i}\}$, where $x_k^{t_i}$ denotes the result when applying t_i to x_k.

Now, imagine that we have a pre-trained encoder f_{enc}, and we pass every images in each set through this encoder. As a result, we obtain three sets of vectors: Z from the original images X, and Z^{t_1}, Z^{t_2} from the augmented images X^{t_1}, X^{t_2}. Depending on the effect each operation had on the original images, we might find ourselves in a situation where the similarity between vectors in each set is different. At this point, we want to determine which operation to favor based on their effect on similarity between vectors. We argue that we would want to give a higher score to operations with the highest effects on the similarities between vectors, and we justify this with the following reasoning.

Let us consider a scenario where, based on the vectors obtained from the original images X, x_1 is closer to x_2 than x_3. However, upon applying the augmentation operation t_1, the resulting vectors show that $x_3^{t_1}$ is closer to $x_2^{t_1}$ than $x_1^{t_1}$. Conversely, when t_2 is applied, the structure remains similar to X, where $x_1^{t_2}$ is closer to $x_2^{t_2}$ than $x^{t_2}3$. We argue that t_1 should receive a higher score from our function compared to t_2 because it induces the most significant change in the similarity structure of the vectors. By prioritizing t_1, the aim is for the currently trained encoder f_{enc} to become invariant to t_1, thereby aligning the similarity structure of vectors from X^{t_1} with those from X and X^{t_2}.

Now, we will delve into the details of how we construct f_{rank}. Building upon the intuition provided above, we contend that our scoring function f_{rank} should be crafted to evaluate the **difference** between the similarity structures of two sets of vectors obtained from the original images and the augmented ones. Moreover, it should assign a higher score to operations that produce the most notable **difference** in the similarity structure. Consequently, to represent f_{rank} mathematically, we consider the following requirements:

1. A quantifiable metric for **difference**.
2. A compatible mathematical representation for the similarity structure of a set of vectors.

We propose the following design choice for each requirement:

Quantifying Difference

We choose the Kullback-Leibler (KL) Divergence as the metric for f_{rank}. The KL Divergence measures the deviation of one probability distribution from another, where the latter acts as the reference [Csiszar 1975]. This metric provides a suitable mathematical framework to quantify the disparity between the similarity structures of the augmented vectors and the original vectors, with the latter acting as the reference probability distribution in our scenario. Notably, this choice implicitly treats the similarity structures of vectors as probability distributions, necessitating us to express the similarity structure of a set of vectors mathematically as a probability distribution. We elaborate on this process below.

Expressing Similarity Structure Mathematically

It is important to note that various methods exist for quantifying the similarity between two vectors. For instance, the cosine similarity metric is commonly employed for this purpose. However, our focus extends beyond the similarity between individual vectors to encompass the broader similarity structure inherent in a set of vectors. This mathematical quantity should contain enough information about the relative positions of the vectors to allow for comparison. Since we choose KL-Divergence as our comparison metric, this mathematical expression of similarity structure must be represented as a distribution.

To address this challenge, we propose the following approach. Starting with a set of vectors denoted as $Z = \{z_1, z_2, \ldots, z_n\}$ obtained from the pre-trained encoder f_{enc}, we begin by computing the pairwise similarities between each vector in the set. This computation yields a similarity matrix, denoted as $\mathbf{S} \in \mathbb{R}^{n \times n}$, where $\mathbf{S}_{ij}$ represents the similarity between z_i and z_j. Next, we normalize this matrix by applying a softmax operation along its rows. This normalization process transforms each row $\mathbf{S}_i$ into a set of categorical probabilities, where each element within the row represents the probability of the corresponding vector z_i being similar other vectors in the set Z. In other words, after applying the softmax operation to the similarity matrix $\mathbf{S}$ along its rows, we obtain $\mathbf{P}$ where $\mathbf{P}_{ij}$ denotes the probability of z_i being similar to z_j:

$$\mathbf{P}_{ij} = \frac{e^{\mathbf{S}_{ij}}}{\sum_{k=1}^{n} e^{\mathbf{S}_{ik}}}, \tag{4.1}$$

where:

- n: Number of elements in the set,

- $\mathbf{S}_{ij} \in \mathbb{R}$: Similarity between vectors z_i and z_j,
- $\mathbf{P}_{ij} \in [0, 1]$: Probability of z_i being similar to z_j.

Note that we use the cosine similarity metric to compute the similarity between vectors such that,

$$\mathbf{S}_{ij} = \frac{z_i \cdot z_j}{\|z_i\| \cdot \|z_j\|}. \tag{4.2}$$

The end result of this process is a matrix $\mathbf{P}$, where each row of the matrix represents the probabilities of each vector in the set being similar to other vectors in the same set.

KL-Divergence Based Scoring Function

Now that we have fufill the requirements for f_{rank}, we can now define the scoring function. Remember that we are given a downstream task d_{task}, an augmentation operation t and the currently trained encoder f_{enc}, to do the ranking. We denote the set of images from the downstream task as $X^{d_{\text{task}}} = \{x_1, x_2, \ldots, x_n\}$. We then apply the augmentation operation t to each image in the set to obtain a new set of images $X^t = \{t(x_1), t(x_2), \ldots, t(x_n)\}$. We then pass both sets of images through the encoder to obtain their vector representations $Z^{d_{\text{task}}} = \{f_{\text{enc}}(x_1), f_{\text{enc}}(x_2), \ldots, f_{\text{enc}}(x_n)\}$ and $Z^t = \{f_{\text{enc}}(t(x_1)), f_{\text{enc}}(t(x_2)), \ldots, f_{\text{enc}}(t(x_n))\}$ respectively. We compute the similarity matrices $\mathbf{S}^{d_{\text{task}}}$ and $\mathbf{S}^t$ for the original and augmented vectors respectively. Then, we normalize these matrices to obtain the probability matrices $\mathbf{P}^{d_{\text{task}}}$ and $\mathbf{P}^t$ respectively. Finally, we compute the KL-Divergence between these two probability matrices to obtain a score for the operation t as shown in Equation 4.3.

$$D_{\text{KL}}\left(\mathbf{P}^{d_{\text{task}}} \middle\| \mathbf{P}^t\right), \tag{4.3}$$

Where:

- n: Number of elements in the set,
- $\mathbf{P}^{d_{\text{task}}} \in \mathbb{R}^{n \times n}$: Probability matrix of the original representations,
- $\mathbf{P}^t \in \mathbb{R}^{n \times n}$: Probability matrix of the augmented representations.

However, it is important to note that in this ranking method, we consider our downstream task as a classification task. Consequently, we assume that d_{rank} also includes a set of class labels $Y^{d_{\text{task}}} = \{y_1, y_2, \ldots, y_n\}$ along with the associated set of images $X^{d_{\text{rank}}}$.

We opt to design our scoring function in such a way that it exclusively focuses on the difference in the similarity structure between images of the same class. This decision was made because analyzing the similarity structure between images from different classes would require a deeper understanding of the semantic content across various classes, which falls beyond the scope of our current work. Our primary aim is to propose

a starting point from which more sophisticated scoring functions can be developed. The effectiveness of our proposed scoring function will be empirically evaluated in Chapter 6.

Since we are concentrating solely on the similarity structure between images of the same class, our final f_{rank} function is defined as follows:

$$f_{\text{rank}}(t, f_{\text{enc}}, d_{\text{task}}) = \sum_{c=1}^{C} D_{\text{KL}} \left(\mathbf{P}^{c,d_{\text{task}}} \middle\| \mathbf{P}^{c,t} \right) \tag{4.4}$$

where:

- C: Number of classes in the downstream task,
- $\mathbf{P}^{c,d_{\text{task}}} \in \mathbb{R}^{n \times n}$: Probability matrix of the original representations with class c,
- $\mathbf{P}^{c,t} \in \mathbb{R}^{n \times n}$: Probability matrix of the augmented representations with class c.

4.3.2 Update Mechanism

The update mechanism follows the ranking mechanism once a probability score is obtained for each operation in our augmentation bag $\mathcal{T}$, as detailed in Algorithm 4. We offer flexibility in the design of an update mechanism, provided it only modifies the augmentation pipeline of the SSL pre-training objective. This process involves constructing a k-stage augmentation pipeline given an augmentation bag $\mathcal{T}$ and its associated probabilities over the operations in the bag. In this context, our **augmentation pipeline** is a sequence of k operations applied to a single image to generate a transformed version of the image. This augmentation pipeline will then be used during the subsequent learning steps of the model in the pre-training phase until another update interval is reached, triggering another rank and update process. We propose two types of update mechanisms: one that selects the top-k operations based on their probabilities and constructs a constant pipeline based on the selected operations, and another that is more dynamic, sampling operations based on their probabilities each time an image needs to be augmented. We provide detailed explanations of each method in the subsequent sections.

Update with Top-K Operations

Upon obtaining the probabilities over the set of operations $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ from any ranking mechanism, as detailed in Section 4.3.1, we construct an augmentation pipeline using the top-k operations with the highest probabilities, ordered by decreasing probability value. This approach prioritizes the operations considered most effective based on the scoring function used during ranking. For example, using the scoring function based on KL-Divergence as explained previously, this would select the k operations that led to the highest change in the similarity structure of the vector representations obtained from our encoder.

For instance, consider a scenario where the augmentation bag, $\mathcal{T}$, contains three operations: t_1, t_2, and t_3, with corresponding probabilities of 0.4, 0.5, and 0.1, respectively. If we assume a pipeline size of $k = 2$ (applying two augmentation operations), we will construct the augmentation pipeline by selecting the top two operations based on their probabilities (in this case, t_2 and t_1 due to their higher probabilities).

In this specific case, the resulting image, x_2, after applying this pipeline to the original image, x_0, would be:

$$x_2 = t_1(t_2(x_0))$$

This equation signifies that operation t_2 is applied first to the original image, x_0, followed by operation t_1 being applied to the result of the first operation, $t_1(t_2(x_0))$. This sequential application reflects the order of selection within the pipeline (highest probability first).

Update with Probaility Distribution on Operations

The approach based on the top-K operations can result in significant variations in generated views, particularly when the probabilities of operations fluctuate. An alternative strategy is to incorporate a sampling process within the augmentation pipeline. Here, each stage of the augmentation pipeline involves sampling an operation from the augmentation bag $\mathcal{T}$, based on the corresponding probabilities obtained from the ranking process.

This update rule employs a probabilistic approach to construct a dynamic augmentation pipeline that changes every time an image needs to be transformed. Let us denote an augmentation bag of N augmentation operations as $\mathcal{T} = \{t_1, t_2, ..., t_N\}$, where each element t_j represents a specific transformation. Also, we define the probability distribution over the set of operations obtained from the ranking as $P_{\mathcal{T}} = \{p_{t_i} | t_i \in \mathcal{T}\}$, where p_{t_i} denotes the probability of selecting operation t_i. Here is a breakdown of the augmentation process at each stage of the pipeline using the probabilistic update rule:

1. **Operation-Magnitude Selection:** A single sample is drawn from $\mathcal{T}$ based on the probability distribution $P_{\mathcal{T}}$.
2. **Augmentation Application:** The chosen operation t_j is applied to the image, creating a transformed version of the image.

As a result, let us consider a k-stage augmentation pipeline built using this update mechanism. Suppose x_0 is the original image, and $A(x, t_j)$ represents the function that applies operation t_j to image x. Therefore, the augmented image x_k after k stages of this process can be expressed as:

$$x_k = A(x_{k-1}, t_j^{(k)}),$$

Here, $t_j^{(k)}$ represents the operation chosen at stage k based on the single probability distribution $P_{\mathcal{T}}$.

4.4 Summary

In this section, we introduced a novel augmentation method for contrastive SSL with the goal of enhancing the quality of learned representations for various downstream tasks. Our approach integrates a rank-update mechanism that leverages some downstream tasks of interest to guide the construction of an augmentation pipeline during pre-training. We introduced a scoring function used to rank augmentation operations based on the dissimilarity in the similarity structure of the representations obtained from the encoder.

It is important to emphasize that this scoring function relies on labels from the downstream tasks, assuming these labels are obtained from a classification task. Therefore, our method, considering the scoring function proposed, is specifically tailored for classification tasks. Additionally, we proposed two update mechanisms to construct the augmentation pipeline based on the results from the ranking step. The first update mechanism selects the top-k operations based on their probabilities, while the second mechanism samples operations to create a dynamic augmentation pipeline. In the next chapter, we will present the experimental setup used to validate our approach.

Chapter 5

Experimental Setup

This chapter details the experimental setup employed to validate our approach. An overarching view of this setup is provided in Section 5.1. Subsequently, specific details concerning our pre-training process are outlined in Section 5.2. This is followed by an explanation of our downstream task evaluation process in Section 5.4.

5.1 Experimental Procedure

In our experiments, each run follows the steps outlined in Section 2.1.2. Specifically, we train a neural network encoder on an unlabeled dataset using a designated contrastive SSL algorithm and a specific data augmentation method. Subsequently, we assess the quality of the learned representations by extracting the encoder and evaluating its output on a set of downstream tasks. We repeat this entire process for five different random initializations, recording the mean and standard deviation of the model's performance on each downstream task. It is crucial to highlight that the primary variation between various experiments is the data augmentation method utilized during the pre-training phase. Further details on each step are provided in the subsequent sections.

We use a two-tailed Mann-Whitney U test to check for statistical significance when comparing two methods, as this test does not assume a specific underlying distribution of the samples [Mann and Whitney 1947]. We set our p-value threshold at 0.05. Therefore, if the probability that the performances of two methods are different is less than 0.05 according to our test, we reject the null hypothesis and conclude that this difference is statistically significant.

5.2 Pre-Training

The initial step in each experiment involves the pre-training of an encoder using a specific augmentation method and a designated contrastive SSL algorithm, as mentioned earlier. While the same pre-training dataset is utilized for all runs, we evaluate our experiments across various downstream tasks datasets. Additional information about the pre-training dataset can be found in Section 5.2.1, and details about the contrastive

SSL algorithms employed are provided in Section 5.2.2. Hyperparameters used during the pre-training phase are presented in Appendix A.2.

5.2.1 Pre-Training Dataset

For this study, we pre-train our models on TinyImageNet [Le and Yang 2015], which comprises 100,000 natural images belonging to 200 categories, ranging from animals, objects, places to people. Since we are utilizing this dataset in an unlabeled setting, we remove the labels attached to the samples. We acknowledge that this dataset is significantly smaller than those predominantly used in the literature, such as ImageNet [Russakovsky *et al.* 2015]. Our decision to use TinyImageNet over the large-scale benchmark dataset ImageNet is primarily motivated by the computational budget required for training on the 14 million images that constitute ImageNet. More details about TinyImageNet are provided in Appendix A.1.

5.2.2 Contrastive SSL Methods

We use two contrastive SSL methods for this study, namely, SimCLR [Chen *et al.* 2020b] and Barlow Twins [Zbontar *et al.* 2021].

SimCLR

SimCLR excels in contrastive visual representation learning due to its effective composition of data augmentation operations and use of a learnable non-linear transformation before the SSL contrastive objective. Another important contribution of Chen *et al.* [2020b] is the utilization of an unsupervised contrastive loss objective on normalized embeddings of images from a single batch of data (as shown in Equation 5.1).

$$\mathcal{L}_{\text{SimCLR}} = -\frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{2N}\log\frac{\exp(\mathbf{S}(z_i,z_j)/\tau)}{\sum_{k=1}^{2N}\exp(\mathbf{S}(z_i,z_k)/\tau)}, \tag{5.1}$$

where:

- N is the number of samples in the batch.
- z_i and z_j are the representations of two augmentations of the same image.
- z_k is the representation of an augmentation of a different image of the same batch.
- $\mathbf{S}(z_i,z_j) = \frac{z_i \cdot z_j}{\|z_i\| \cdot \|z_j\|}$ denotes the cosine similarity between z_i and z_j.
- τ is a temperature parameter that controls the similarity scaling.

The objective of SimCLR is based on a multiclass n-pair loss objective which encourages the representations of views from the same image to be close together while pushing apart those representations if they originate from different images [Sohn 2016]. This objective function is highly dependent on the batch size, as the number of negative samples z_k used in objective function (refer to the denominator in Equation 5.1) is

proportional to the batch size. Consequently, a larger batch size results in more negative samples, influencing the update dynamics during training.

Barlow Twins

The primary distinction between SimCLR and Barlow Twins lies in their unsupervised contrastive learning objectives. In an effort to reduce dependency on batch size, Zbontar *et al.* [2021] proposed a novel contrastive objective that aims to increase the correlation between representations of the same image and decrease this correlation for representations of different images without the need for a large batch size. As evident from the objective function in Equation 5.2, the incentive to increase the correlation between representations of the same image (first term) is completely independent from that of decreasing the correlation between representations of different images (second term).

$$\mathcal{L}_{\text{Barlow Twins}} = \sum_i (1 - \mathcal{C}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} {\mathcal{C}_{ij}}^2 \tag{5.2}$$

In this equation:

- λ is a regularization parameter.
- $\mathcal{C}$ is the cross-correlation matrix computed between the vectors z_i and z_j of the same batch. This matrix is an N×N matrix where N is the dimensionality of z.

$$\mathcal{C}_{ij} = \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}} \tag{5.3}$$

In this equations:

- $z^A_{b,i}$ is the i^{th} dimension of the vetor representation obtained from the augmentation operation A on the b^{th} sample of the batch.

5.3 Baseline Augmentation Methods

This section outlines the data augmentation techniques we will compare our method against. We focus on three specific methods: a fixed augmentation pipeline, RandAugment [Cubuk *et al.* 2020], and TrivialAugment [Müller and Hutter 2021], with a particular focus on the two latter due to their similarities to our approach. Importantly, these two methods share the following key characteristics:

- **Dynamic Augmentation Pipelines:** They dynamically construct unique augmentation sequences for each image on the fly, instead of relying on a fixed set of operations. This dynamic approach allows for a broader range of transformed views to be generated.

- **Non-Learnable Methods:** They do not require training any additional parameters to determine the augmentation pipeline. The entire process is parameter-free, making them computationally efficient.

5.3.1 Fixed Augmentation Pipeline

This augmentation method is the most basic, consisting of a fixed sequence of augmentation operations used throughout pre-training. In this study, we use an augmentation pipeline with a configuration similar to those employed in various contrastive SSL studies [Chen *et al.* 2020b; Tian *et al.* 2020; Zbontar *et al.* 2021]. The pipeline starts with a random cropping operation, followed by a random horizontal flip operation, random color jittering, grayscale conversion, and concludes with gaussian blurring. Comprehensive details regarding the specific parameters for each operation are provided in Appendix A.2.3.

5.3.2 RandAugment

Introduced as an efficient augmentation technique for supervised learning, RandAugment addresses the limitations of heavy hyperparameter optimization methods [Hataya *et al.* 2019; Lim *et al.* 2019; Cubuk *et al.* 2019]. It randomly selects a set of augmentation operations from a predefined set $\mathcal{T}$ and applies them to the input image, similar to our probabilistic update rule (see Section 4.3.2) with the only exception that the probability over operations is random in this case. Despite its simplicity, RandAugment has demonstrated superior performance to methods relying on hyperparameter optimization on supervised image classification benchmarks [Cubuk *et al.* 2020]. We aim to understand how this superiority will translate to our SSL setting.

Let $\mathcal{T} = \{t_1, t_2, ..., t_N\}$ represent the set of N augmentation operations, and $M = \{m_1, m_2, ..., m_P\}$ denoting P magnitudes with which each operation can by applied.

The augmentation process can be summarized as follows:

1. **Operation Selection:** We randomly sample an operation t_j from the set $\mathcal{T}$.
2. **Magnitude Selection:** Once the operation is chosen, we randomly select a magnitude level m_i from the available options for t_j.
3. **Augmentation Application::** The chosen operation t_j is applied to the image with the selected magnitude m_i, resulting in the augmented image.

For instance, let us consider a k-stage augmentation pipeline. Suppose x_0 is the original image, and $A(x, t_j, m_i)$ represent the function that applies operation t_j with magnitude m_i to image x. The augmented image x_k after k stages of this process can be expressed as:

$$x_k = A(x_{k-1}, t_j^{(k)}, m_i^{(k)}), \tag{5.4}$$

where:

- $t_j^{(k)}$ is the operation randomly chosen at stage k,
- $m_i^{(k)}$ is the magnitude level randomly chosen for operation $t_j^{(k)}$ at stage k.

By pre-defining the augmentation operations and their magnitudes, RandAugment avoids the need for extensive hyperparameter search, making it an efficient and effective data augmentation method. However, it is important to consider that the specific choice of operations and magnitudes in $\mathcal{T}$ and M can influence the effectiveness of RandAugment for a particular task.

In our experiments, we set $k = 2$ operations (as in Section 6.2.1) using the same operations and parameters as described in the paper. It is worth noting that this setup naturally results in a richer set of augmented views compared to our method, as the paper employs more operations in its augmentation bag $\mathcal{T}$ than our framework.

5.3.3 TrivialAugment

This augmentation method was proposed with the same motivation as RandAugment, aiming to provide a more efficient augmentation technique for supervised learning without the need for any computationally expensive tuning process [Müller and Hutter 2021]. The main difference from RandAugment is that it employs a single operation in its pipeline ($k = 1$) and considers a broader range of magnitudes m_i for each operation. For this experiment, we adopt the same setup as described in the paper. TrivialAugment can be viewed as a special case of RandAugment where $k = 1$.

TrivialAugment simplifies the augmentation process by applying only one randomly chosen operation from a predefined set $\mathcal{T}$ to the input image. Despite its simplicity, it has been shown to achieve performance comparable to that of RandAugment [Müller and Hutter 2021], with the obvious improvement in speed due to applying only one augmentation operation to an image.

5.4 Evaluation on Downstream Tasks

Once the pre-training phase is completed, we extract the pre-trained encoder f_{enc} from the previous training pipeline and evaluate the quality of the learned representations on a set of classification tasks. We provide more details about the datasets used for evaluation in Section 5.4.1 and the evaluation pipeline in Section 5.4.2. Hyperparameters used in this phase are outlined in Appendix A.2.

5.4.1 Evaluation Datasets

Our primary focus is to evaluate performance on classification datasets. We make an attempt to assess models across diverse types of datasets. This exploration stems from our interest in understanding how learned representations transfer to different domains. Consequently, our dataset comprises primarily natural image datasets, including CIFAR100 [Krizhevsky and Hinton 2009], DTD [Cimpoi *et al.* 2014], Flowers102 [Nilsback and Zisserman 2008], Pets [Parkhi *et al.* 2012], and the training split of STL10

[Coates *et al.* 2011]. Beyond these natural images, we extend our evaluation to include satellite imagery, specifically RESISC45 [Cheng *et al.* 2017]. We provide more details about these datasets in Appendix A.1.

5.4.2 Evaluation Pipeline

We evaluate the quality of the learned representations using two methods: *Linear Evaluation* and *K-Nearest Neighbors*. Both methods are detailed in auto 2.1.2. We chose these methods over fine-tuning because we want to evaluate the linear seperability of the learned representations. In other words, we want to evaluate how well the learned representations can be used to classify images without further refinement of the learned representation space. Note that we use top-1 accuracy as our metric to measure performance for both methods. Details about the hyperparamrters used in each method are provided in Appendix A.2.

Linear Evaluation

In this method, we use the learned representations as features for a linear classifier. Then, we train the classifier on the training split of each datasets outlined in Section 5.4.1 and evaluate the performance of the classifier on the test split.

K-Nearest Neighbors

In addition to training a linear classifier, we fit a KNN classifier on the training of split of each datasets. We then evaluate the performance of the KNN classifier on the test split of the same datasets.

5.5 Summary

In this chapter, we outlined the overall setup for our SSL experiments. We discussed the datasets, our baselines, and detailed the evaluation process designed to assess the quality of the learned representations. Following this, in Chapter 6, we present the outcomes of experiments conducted using this setup and provide an analysis of the findings.

Chapter 6

Results & Analysis

Having described our Rank-Update Framework in Chapter 4 and detailed our experimental setup in Chapter 5, we now present our experimental results and provide a detailed analysis of our findings. We outline the different experiments we plan to conduct to validate the usefulness of our framework. Our experiments are categorized into three sections, as outlined in the following paragraphs but we also provide an outline of these experiments in Table 6.1.

Table 6.1: Table of Experiments

Experiment	Description
Understanding Data Augmentation in Contrastive SSL	
Choice of Operations	Investigates which specific augmentation operations are most beneficial for contrastive SSL (Section 6.1.1).
Order of Operations	Evaluates the impact of the order in which augmentation operations are applied within the pipeline (Section 6.1.2).
Compare Rank-Update against Known Baseline Augmentation Methods	
Rank-Update vs Constant	Compares the Rank-Update framework to a baseline where the same augmentation pipeline is used for all downstream tasks (Section 6.2.1).
Rank-Update vs Parameter-Free	Compares the Rank-Update framework against other augmentation methods similar to ours (Section 6.2.2).
Evaluate Rank-Update Accross Different Configurations	
Rank-Update configurations	Analyzes the impact of different configurations of the Rank-Update framework on performance (Section 6.3.1).
SSL configurations	Investigates how changes to the overall contrastive SSL configuration influence performance (Section 6.3.2).

Initially, we focus on understanding how the augmentation pipeline influences performance in contrastive SSL. This investigation is crucial to motivate our work and justifying the design decisions made for our framework. We delve into how the selection of operations impacts performance in Section 6.1.1. Additionally, we expand on this by examining the impact of the order of operations in the augmentation pipeline, as detailed in Section 6.1.2.

Having sufficient motivation to propose improvements to the data augmentation pipeline in contrastive SSL in Section 6.1, our subsequent objective is to examine how our proposed approach compares to other data augmentation methods. This assessment aids in gauging the novelty and efficacy of our contribution to the existing literature. We begin by comparing our method against a fixed augmentation pipeline in Section 6.2.1 and subsequently evaluate its performance against other non-learnable augmentation methods in Section 6.2.2.

Finally, we investigate how different training configurations influence our framework. This exploration is vital for assessing the stability and practicality of our method. In Section 6.3, we analyze how the performance of our method is affected by various learning hyperparameters, both specific to our method (see Section 6.3.1) and to the pre-training process in general (see Section 6.3.2). This provides insights into the sensitivity of our method to various SSL configurations.

6.1 The Augmentation Pipeline in Contrastive SSL

In this section, our aim is to evaluate the impact of a fixed (constant) augmentation pipeline on the performance of contrastive SSL algorithms. To achieve this, we conduct two sets of experiments to investigate how the selection and sequencing of augmentation operations within the pipeline affect the performance of contrastive SSL algorithms. It is important to note that we do not conduct any experiments on the Rank-Update framework in this section. This is because our objective here is to provide a comprehensive understanding of the significance of data augmentation in contrastive SSL, as commonly used in current literature.

6.1.1 Choice of the Augmentation Operations

In this experiment, our goal is to empirically demonstrate that the careful selection of augmentation operations for pre-training significantly impacts the performance of contrastive SSL. We aim to broaden the evaluation scope by considering multiple downstream tasks beyond the traditional focus on a few ImageNet-like classification tasks, as seen in previous studies [Tian *et al.* 2020; Chen *et al.* 2020b]. Additionally, we consider the influence of another contrastive SSL algorithm (Barlow Twins), whose objective function does not explicitly rely on pushing apart negatives [Zbontar *et al.* 2021]. By doing this, we also seek to understand how the explicit reliance on negative sample repulsion by popular SSL algorithms such as SimCLR affects results when compared to algorithms that do not rely on it like Barlow Twins.

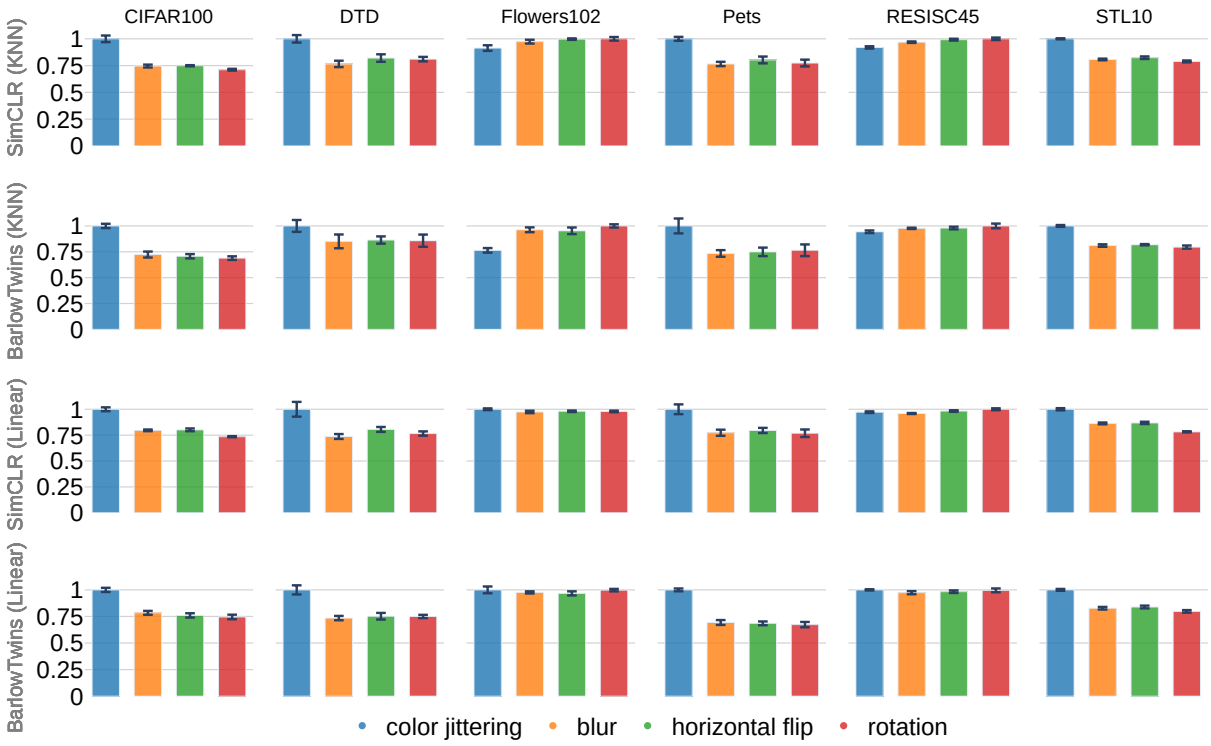


Figure 6.1: Average performance of SSL algorithms trained with four augmentation operations separately and evaluated on all six downstream tasks. For each bar plot, we normalize the results of the 4 operations with respect to the best performance in that plot to focus on the trends along each downstream task.

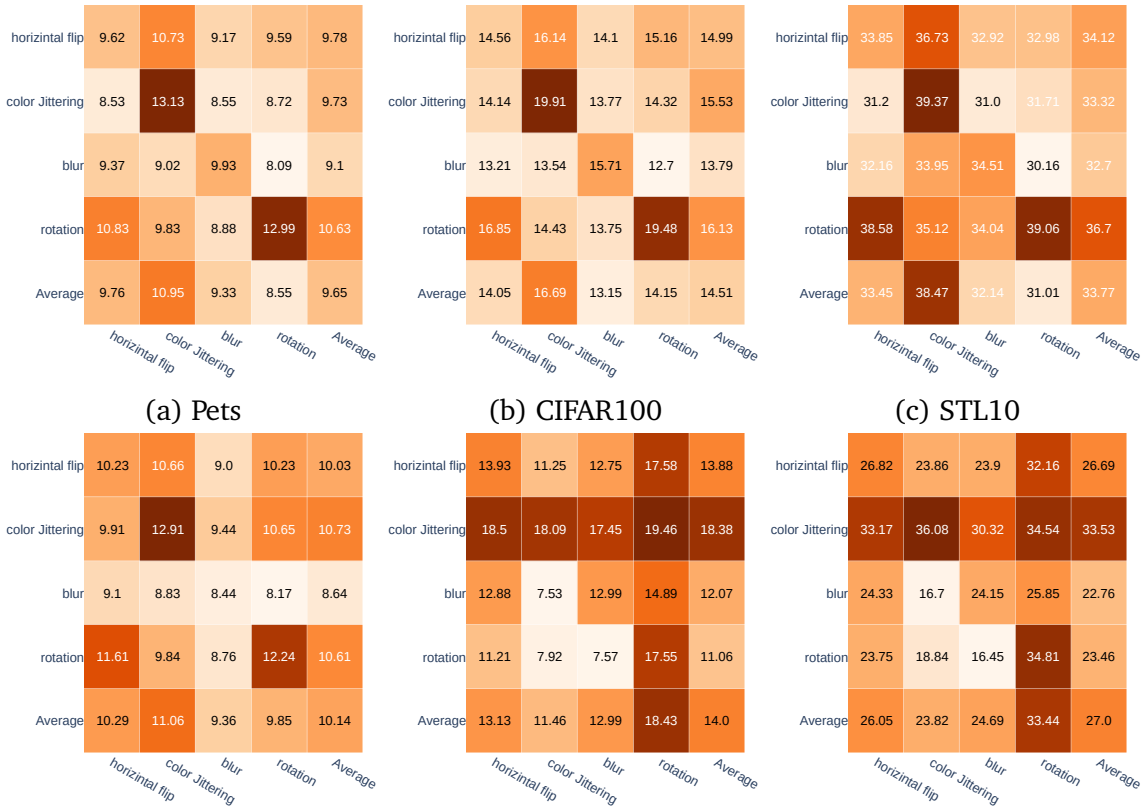
We evaluate the performance of SimCLR and Barlow Twins models, which are pre-trained according to Section 5.2, and assess them on six downstream tasks as described in Section 5.4. This evaluation is conducted using four different augmentation pipelines. Each pipeline includes an image resizing operation followed by a single augmentation operation. We deliberately avoid cropping, a technique commonly used in contrastive SSL pipelines. This choice allows us to isolate the effect of the specific augmentation operation that follows the resize operation. The four augmentation operations used are **color jittering**, **blur**, **horizontal flip**, and **random rotation**. We select these operations due to the diverse range of properties they modify in an image, from color to orientation. The objective of this experiment is to observe how each operation affects the SSL algorithm's performance for different downstream tasks. We only allow one operation after the resize operation to further isolate the effect of each augmentation operation. More complex scenarios involving multiple operations are examined in Section 6.1.2.

The results presented in Figure 6.1 confirm previous observations made in contrastive SSL [Tian *et al.* 2020; Chen *et al.* 2020b; Wagner *et al.* 2022], highlighting the significant impact of augmentation operations on contrastive SSL performance. This impact is evident when examining the performance distribution across different operations on various downstream tasks. For example, **color jittering** proves effective for CIFAR100

and STL10 but falls short as the optimal choice for Flowers102. This suggests that operations promoting invariance to color should be avoided, especially considering that many flowers are distinguished by their colors. Furthermore, a consistent distribution is observed across our two SSL algorithms (along the columns), indicating that in our setup, the relevance of an operation depends more on the downstream task than on the specific SSL algorithm being used.

6.1.2 Order of the Augmentation Operations

In Section 6.1.1, we illustrated how augmentation operations affect the performance of contrastive SSL algorithms. Now, we delve into a more intricate scenario. Considering that an augmentation pipeline consists of multiple operations sequenced together, our goal is to investigate the impact of combining these diverse operations within a single pipeline. We aim to assess how the relevance of an augmentation operation to a downstream task (as observed in Section 6.1.1) is influenced by other operations in the same augmentation pipeline.



(a) Pets (b) CIFAR100 (c) STL10

(d) DTD (e) Flowers102 (f) RESISC45

Figure 6.2: The performance of SimCLR with linear evaluation on various downstream tasks is shown here for different 2-stage data augmentation pipelines. In each heatmap, rows represent the first augmentation operation applied, and columns represent the second. The opacity of a cell's color is directly proportional to the average accuracy achieved on the corresponding downstream task

We construct 16 unique two-stage augmentation pipelines, each comprising permutations of any two of the four augmentation operations used in Section 6.1.1. Consequently, some pipelines involve the repetition of the same operation twice, while others feature the same pairs of operations in reverse order. The training process closely follows that of the previous experiment with diverse augmentation pipelines. However, due to the computational demands associated with training numerous models using various augmentation pipelines, we choose to focus exclusively on SimCLR in this experiment. Nevertheless, the consistent performance observed across different SSL algorithms, including Barlow Twins and SimCLR, in Section 6.1.1, suggests that these findings might be applicable to similar methods. Future research should explicitly explore this to validate our assumption.

Figure 6.2 displays heatmaps for various downstream tasks. Each cell represents the average accuracy achieved by SimCLR with linear evaluation on a specific task using a particular data augmentation pipeline (the order is indicated by the axes). Rows represent the first applied operation, while columns represent the second. The color opacity reflects the average accuracy for the corresponding augmentation pipeline. For example, the cell in the first row and second column shows the performance when horizontal flip is applied first, followed by color jittering. The last row and column show the average performance of each operation across all pipelines where it appears first and second, respectively.

Upon examining the average performance of each operation across different positions in the pipeline, it becomes evident that the order of operations significantly impacts the model's performance. For instance, the **rotation** operation exhibits higher average performance when placed as the first operation in the pipeline for Pets, CIFAR100, and STL10. However, its average performance decreases when it is positioned as the second operation in the pipeline. Additionally, we observe consistent performance of an operation across downstream tasks that share similar distributions of performance over augmentation operations, as seen in Section 6.1.1. For example, the heatmap of RESIS45 resembles Flowers102's heatmap more than others. Notably, in Section 6.1.1, RESISC45 and Flowers102 were the only tasks where **rotation** yielded the highest performance. This observation suggests that RESISC45 and Flowers102 may share some underlying characteristics that lead them to favor the same set of augmentation operations. Lastly, we notice that two-stage pipelines constructed with repeating augmentation operations provide the most benefits to performance. This is evident in the case of STL10, CIFAR100, Pets, and DTD for the **rotation** operation, as well as in the case of RESIS45 and Flowers102 for both **rotation** and **color jittering**.

6.2 The Rank-Update Framework Against Other Augmentation Methods

The results discussed in Section 6.1 emphasize the importance of selecting appropriate augmentation operations for effective performance in contrastive SSL. Building upon this, we explore ways to enhance this process with our proposed framework. We begin by comparing our method with the widely used fixed augmentation pipeline outlined

in Section 6.2.1, commonly found in contrastive SSL literature [Chen *et al.* 2020b; Caron *et al.* 2018; Grill *et al.* 2020]. Subsequently, in Section 6.2.2, we assess the performance of our method against other non-learnable augmentation methods. This analysis provides insights into how our proposed framework compares with alternative augmentation approaches.

6.2.1 Performance Against a Fixed Augmentation Pipeline

In the following sections, we undertake a comparative analysis, contrasting a fixed augmentation method as described in Section 5.3.1 with our proposed framework, which uses a more complex way for performing data augmentation in contrastive SSL.

KL-Div Ranking + Top-K Update Rule

We begin by assessing the effectiveness of our rank-update method, as elaborated in Section 4.3, in comparison to a fixed augmentation pipeline, as described in Section 5.3.1. Our evaluation starts with the top-K update rule outlined in Section 4.3.2. Recognizing the importance of random cropping in contrastive SSL [Chen *et al.* 2020b], we designate it as the initial operation in any pipeline generated through our method. Subsequently, following this cropping stage, we incorporate the top-ranked operations in decreasing order of probability, as per the top-K update rule. It is important to clarify that we utilize all classification tasks outlined in Section 6.1 for ranking, while also individually evaluating the learned representations for each of these tasks. This approach ensures that the ranking mechanism in our rank-update method remains unbiased towards any specific downstream task and instead aims to derive an augmentation pipeline that is generally beneficial across different tasks on average.

We update the augmentation pipeline using the rank-update method every 10 epochs, denoted by the update interval $E = 10$, and sample 200 samples per downstream task of interest for this purpose. Additionally, we implement our Top-K update rule with $K = 2$, meaning that we include the top-2 operations with the highest probabilities, according to the ranking, into the augmentation pipeline. This results in a 2-stage augmentation pipeline, with random cropping as the initial operation. This decision is based on previous research demonstrating the effectiveness of random cropping across various contrastive SSL methods [Chen *et al.* 2020b]. Therefore, we set random cropping as the first stage of our augmentation pipeline without prior ranking. We explore different configurations of our method in Section 6.3.1.

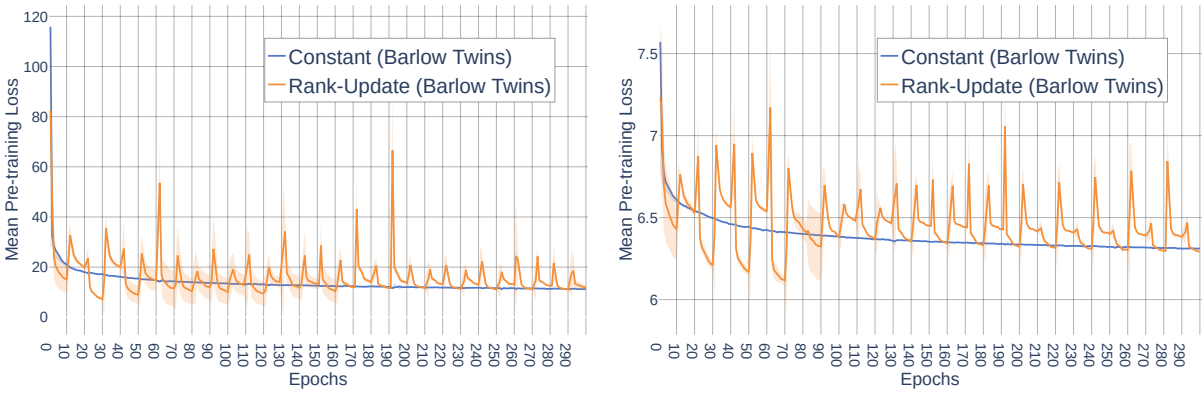
The results presented in Table 6.2 and Table 6.3 indicate that our method exhibits lower performance across all downstream tasks, leading to an overall decrease in average performance. Additionally, a notable standard deviation in the results highlights significant instability, emphasizing the model's sensitivity to changes in the augmentation pipeline. Specifically, Figure 6.3 showcases substantial fluctuations in the loss values during pre-training, with spikes occurring every 10 epochs, corresponding to the frequency of the rank-update step ($E = 10$). Further investigation of the pre-training phase, especially the KNN accuracy evolution for each downstream task (refer to Figure 6.4), reveals a similar pattern.

Table 6.2: Top-1 Accuracy with linear evaluation on downstream tasks for our method, ranked based on KL-Divergence and updated using the Top-K update rule ($K=2$). Note that $*$ means *non-statistically significant difference*

Downstream Tasks	SimCLR	SimCLR (RU)	Barlow	Barlow (RU)
CIFAR100	35.46 ± 0.41	20.90 ± 0.89 ↓	28.76 ± 0.33	21.79 ± 2.00 ↓
STL10	57.02 ± 0.37	44.13 ± 0.69 ↓	51.82 ± 0.29	43.43 ± 2.82 ↓
Pets	21.03 ± 0.44	14.91 ± 0.34 ↓	15.80 ± 0.56	12.50 ± 0.70 ↓
DTD	23.54 ± 0.76	17.54 ± 0.57 ↓	20.83 ± 0.43	17.92 ± 2.27 ↓
RESISC45	48.25 ± 0.64	42.67 ± 0.95 ↓	41.02 ± 0.59	37.30 ± 2.61 ↓
Flowers102	22.39 ± 0.29	16.71 ± 0.60 ↓	16.04 ± 0.24	16.05 ± 2.18 $\star$
Average Performance	34.62 ± 0.49	26.15 ± 0.67 ↓	29.04 ± 0.41	24.83 ± 2.10 ↓

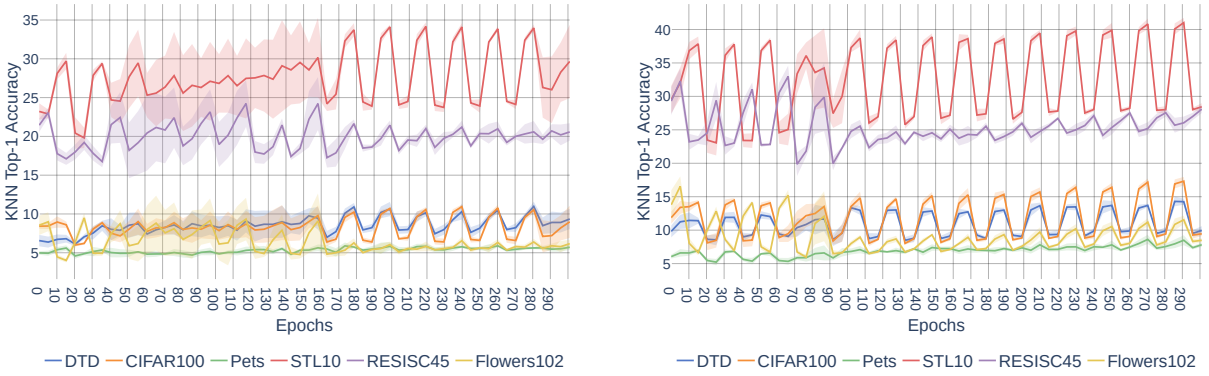
Table 6.3: Top-1 Accuracy with KKN evaluation on downstream tasks for our method, ranked based on KL-Divergence and updated using the Top-K update rule ($K=2$).

Downstream Tasks	SimCLR	SimCLR (RU)	Barlow	Barlow (RU)
CIFAR100	26.02 ± 0.13	09.72 ± 0.33 ↓	17.23 ± 0.42	09.62 ± 1.74 ↓
STL10	48.22 ± 0.32	28.13 ± 0.82 ↓	40.22 ± 0.43	29.36 ± 4.57 ↓
Pets	11.35 ± 0.47	07.90 ± 0.49 ↓	07.86 ± 0.34	05.43 ± 0.30 ↓
DTD	14.77 ± 0.30	09.82 ± 0.29 ↓	12.03 ± 0.56	10.19 ± 1.57 ↓
RESISC45	31.92 ± 0.48	28.77 ± 0.82 ↓	22.92 ± 0.88	20.90 ± 0.76 ↓
Flowers102	09.16 ± 0.20	08.03 ± 0.45 ↓	04.25 ± 0.32	05.34 ± 0.22 ↑
Average Performance	23.57 ± 0.31	15.39 ± 0.53 ↓	17.42 ± 0.49	13.48 ± 1.51 ↓



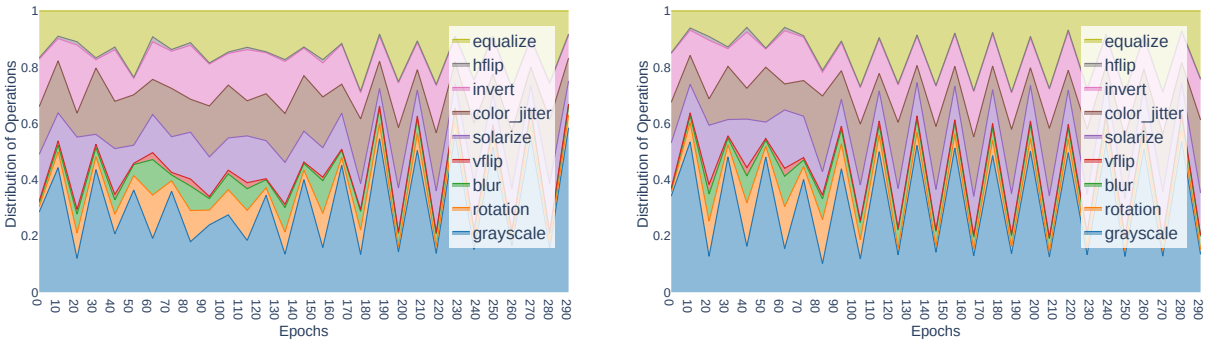
(a) Barlow Twins (b) SimCLR

Figure 6.3: Evolution of the pre-training loss for Barlow Twins and SimCLR with a fixed augmentation pipeline compared to our method.



(a) Barlow Twins (b) SimCLR

Figure 6.4: Evolution of the KNN accuracy for Barlow Twins with our method over different downstream tasks.



(a) Barlow Twins (b) SimCLR

Figure 6.5: Stacked area plot of the evolution probabilities of different operations in the augmentation bag $\mathcal{T}$ throughout pre-training.

We hypothesize that this behavior stems from the model's tendency to overfit to the specific augmentation pipeline constructed during the limited epochs interval in which this pipeline is utilized to generate new views. As a result, the model becomes increasingly invariant to the operations within this pipeline. If this pattern repeats every 10 epochs, the model undergoes readaptation to each newly constructed augmentation pipeline, which in turn triggers a spike in the loss value. This observation is further supported by the evolution of the probability distribution during pre-training in Figure 6.5, where the distribution over operations oscillates between *Equalize* and *Grayscale*. This suggests a continual learning issue wherein the model exhibits overfitting tendencies to the currently selected augmentation operations, potentially leading to catastrophic forgetting of the representations learned from the previously selected operations when the augmentation pipeline is updated.

KL-Div Ranking + Probabilitic Update Rule

In this section, we address the instability observed earlier by focusing on the problem of the pre-trained model becoming too specialized to a particular augmentation pipeline

generated using the Top-K update rule (Section 4.3.2). This update rule selects operations based on their probabilities, forming a fixed augmentation pipeline. However, this can lead to the model forgetting previously learned representations if the probabilities of operations shift significantly. To mitigate this issue, we replace the Top-K update rule with the Probabilistic update rule (Section 4.3.2). The Probabilistic update rule introduces randomness by sampling operations from a probability distribution. This ensures that even less probable operations have a chance to be included in the augmentation pipeline, promoting diversity and adaptability.

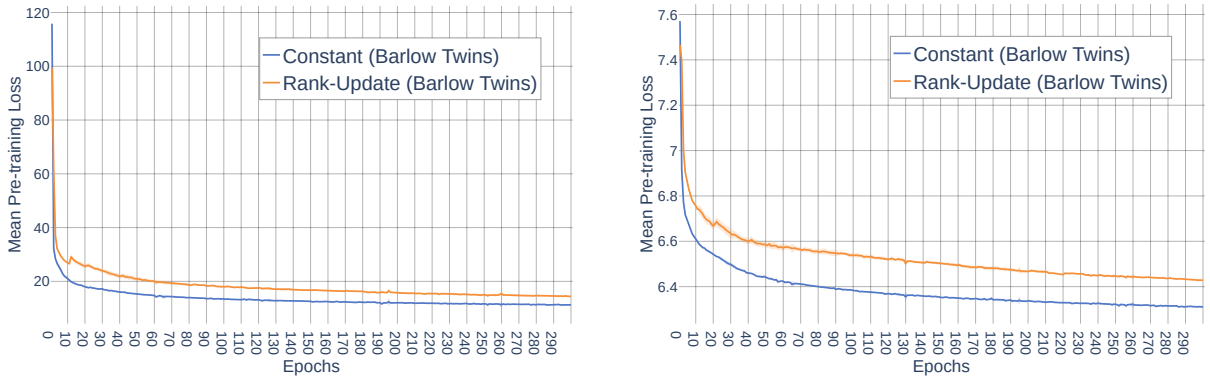
Table 6.4: Top-1 Accuracy with linear evaluation on downstream tasks for our method, ranking based on KL-Divergence and update using the Probabilistic update rule (K=2). Note that $*$ means *non-statistically significant difference*

Downstream Tasks	**SimCLR**	**SimCLR (RU)**	**Barlow**	**Barlow (RU)**
CIFAR100	35.46 ± 0.41	36.85 ± 0.16 ↑	28.76 ± 0.33	30.89 ± 0.29 ↑
STL10	57.02 ± 0.37	56.02 ± 0.27 ↓	51.82 ± 0.29	52.01 ± 0.31 ⋆
Pets	21.03 ± 0.44	22.00 ± 0.68 ⋆	15.80 ± 0.56	18.23 ± 0.42 ↑
DTD	23.54 ± 0.76	28.01 ± 0.61 ↑	20.83 ± 0.43	25.45 ± 0.77 ↑
RESISC45	48.25 ± 0.64	61.92 ± 0.35 ↑	41.02 ± 0.59	52.97 ± 0.65 ↑
Flowers102	22.39 ± 0.29	34.36 ± 0.40 ↑	16.04 ± 0.24	26.27 ± 1.09 ↑
Average Performance	34.62 ± 0.49	39.86 ± 0.41 ↑	29.04 ± 0.41	34.31 ± 0.59 ↑

Table 6.5: Top-1 Accuracy with KNN evaluation on downstream tasks for our method, ranking based on KL-Divergence and update using the Probabilistic update rule (K=2). Note that $*$ means *non-statistically significant difference*

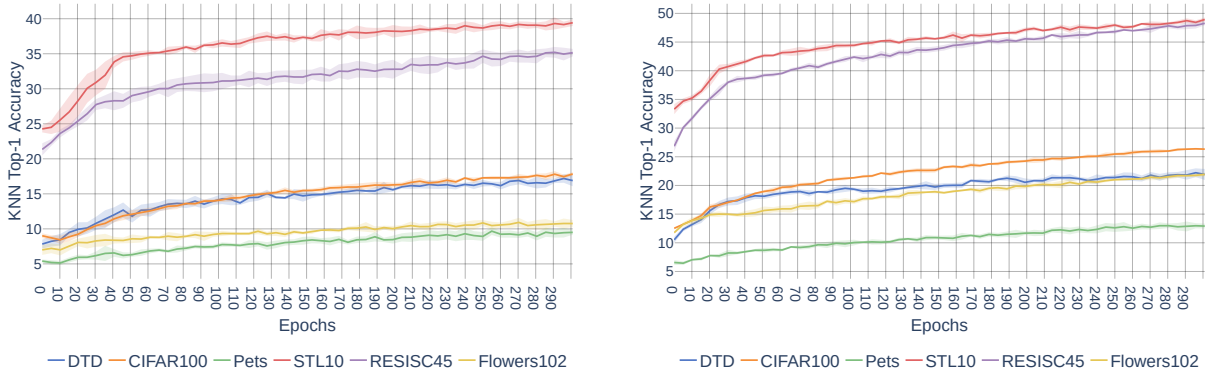
Downstream Tasks	**SimCLR**	**SimCLR (RU)**	**Barlow**	**Barlow (RU)**
CIFAR100	26.02 ± 0.13	25.67 ± 0.18 ↓	17.23 ± 0.42	18.05 ± 0.23 ↑
STL10	48.22 ± 0.32	46.37 ± 0.31 ↓	40.22 ± 0.43	38.93 ± 0.69 ↑
Pets	11.36 ± 0.47	11.86 ± 0.56 ⋆	07.86 ± 0.34	09.02 ± 0.33 ↑
DTD	14.77 ± 0.30	21.30 ± 0.44 ↑	12.03 ± 0.56	15.40 ± 0.60 ↑
RESISC45	31.92 ± 0.48	50.37 ± 0.40 ↑	22.92 ± 0.88	36.56 ± 0.76 ↑
Flowers102	09.16 ± 0.20	17.68 ± 0.10 ↑	04.25 ± 0.32	09.91 ± 0.39 ↑
Average Performance	23.57 ± 0.31	28.87 ± 0.33 ↑	17.42 ± 0.49	21.31 ± 0.50 ↑

The results in Tables 6.4 and 6.5 show notable improvements over those in Section 6.2.1. Probabilities fluctuate slightly, as seen in Figure 6.8, but are less pronounced and stabilize quickly, converging to a relatively constant distribution. While stable loss, depicted in Figures 6.6 and 6.7, is not a direct indicator of accuracy, it points to better training conditions. Interestingly, the operation distribution skews towards *Equalize* and *Grayscale*, which topped the rankings in Section 6.2.1.



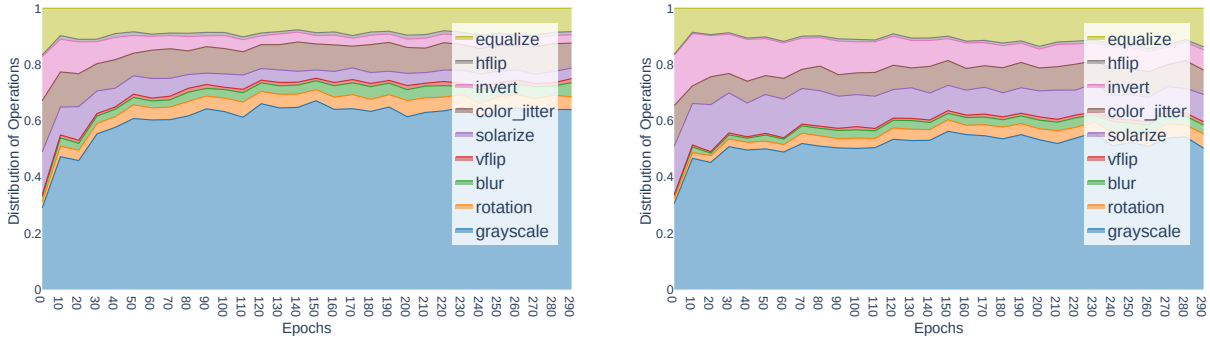
(a) Barlow Twins (b) SimCLR

Figure 6.6: Evolution of the pre-training loss for Barlow Twins and SimCLR with a fixed augmentation pipeline compared to our method.



(a) Barlow Twins (b) SimCLR

Figure 6.7: Evolution of the KNN accuracy for Barlow Twins and SimCLR with our method over different downstream tasks.



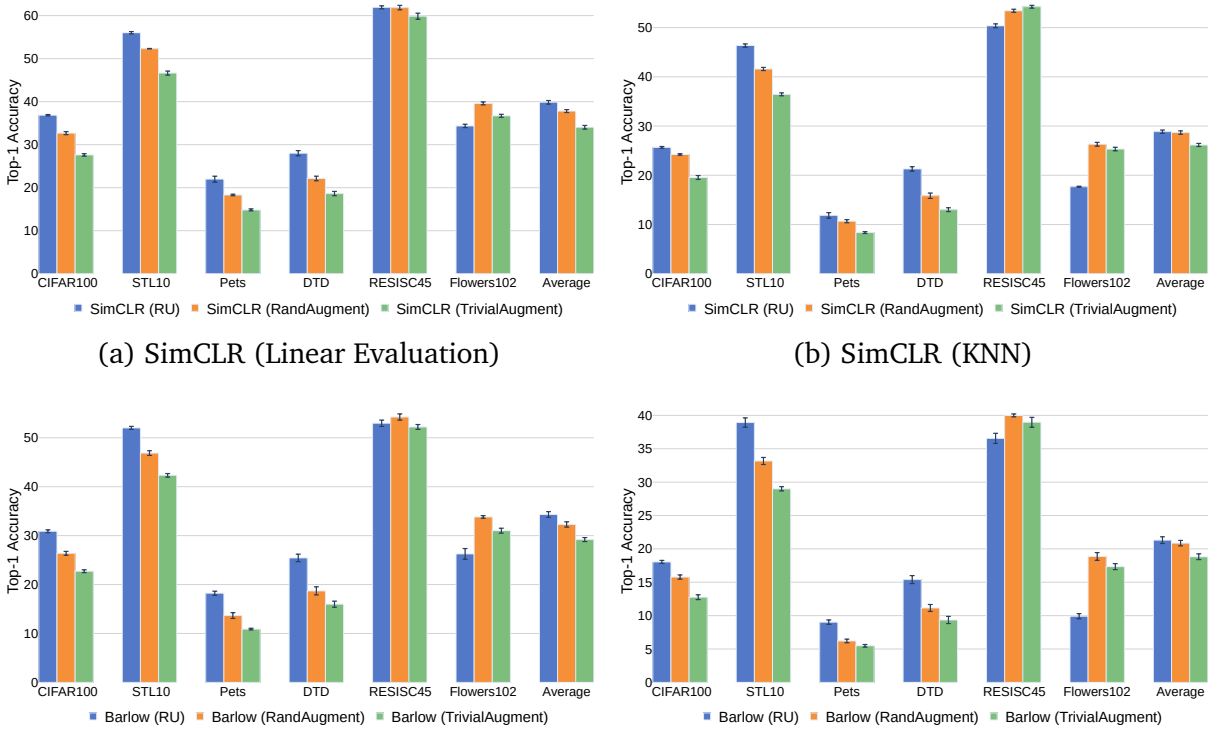
(a) Barlow Twins (b) SimCLR

Figure 6.8: Stacked area plot of the evolution of probabilities for different operations in the augmentation bag $\mathcal{T}$ throughout pre-training.

6.2.2 Performance Against Other Parameter-Free Methods

As our method is parameter-free, meaning that we do not optimize augmentation parameters based on specific quantitative metrics, we compare it to other parameter-free approaches such as RandAugment (see Section 5.3.2) and TrivialAugment (see Section 5.3.3). The results depicted in Figure 6.9 reflect the patterns observed in Table 6.4 and Table 6.5. Our method exhibits superior performance across most downstream tasks, except for Flowers102 and RESISC45, where a significant decline in performance is notable, particularly in the KNN evaluation.

Furthermore, considering the insights from Section 6.1.1, we recall the similar performance trends observed for these two datasets. This observation suggests that our ranking mechanism may prioritize different operations than those optimal for these datasets. We further explore this analysis in Section 6.3.1 by investigating how the choice of downstream tasks influences the ranking mechanism.



(a) SimCLR (Linear Evaluation)

(b) SimCLR (KNN)

(c) Barlow (Linear Evaluation)

(d) Barlow (KNN)

Figure 6.9: Bar plots of top-1 accuracies of SimCLR and Barlow with TrivialAugment, RandAugment and our RU augmentation method.

6.3 Ablations over Different configurations

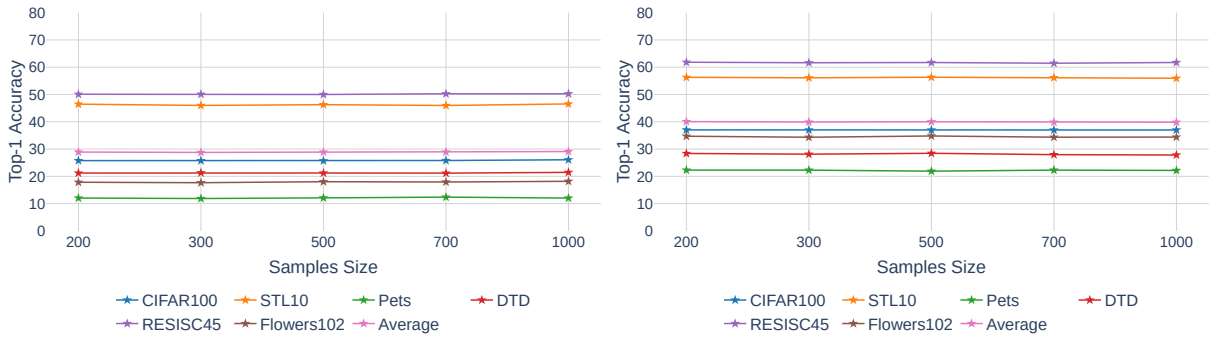
6.3.1 Rank-Update configurations

In this section, we undertake a series of experiments to explore the impact of different configurations of our method. The objective is to gain insights into how each compo-

nent of our approach influences the stability of the pre-training process and its effects on the model's performance across various downstream tasks. The experimental setup aligns with the one outlined in Section 6.2, where we employ the probabilistic update rule elucidated in Section 4.3.2, and the ranking is based on KL-Divergence, as described in Section 4.3.1. However, for this section, we exclusively use SimCLR as our pre-training method, with a focus on identifying trends in the results rather than assessing the model's absolute performance.

Effect of the Samples Size

This section delves into investigating the influence of the number of samples utilized for ranking. This parameter signifies the quantity of samples per downstream task utilized to conduct the ranking process. The objective is to evaluate the extent to which our method relies on the size of labeled data employed for ranking. If a significant amount of labeled data is necessary, it could potentially outweigh the benefits of our method, reintroducing the annotation bottleneck that studies in SSL have strived to overcome.



(a) KNN Evaluation

(b) Linear Evaluation

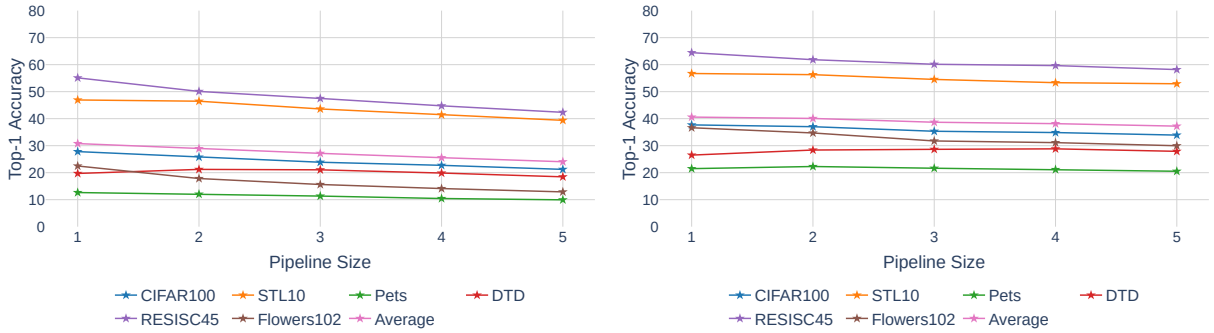
Figure 6.10: Average performance of RU on SimCLR at various samples sizes for different downstream tasks.

In our experiments, we vary the sample sizes across five different values: 200, 300, 500, 700, and 1000, for all downstream tasks to evaluate their impact on the performance of our method. It is important to note that since the downstream tasks have varying numbers of classes, the number of samples per class varies accordingly. The results presented in Figure 6.10 indicate that the performance of our method remains largely unaffected by the number of samples used for ranking. We observe consistent performance across all downstream tasks, regardless of the sample size. This robustness suggests that our method can maintain its effectiveness even with a relatively small number of samples used for ranking, highlighting its practicality in scenarios with limited labeled data.

Effect of the Pipeline Size

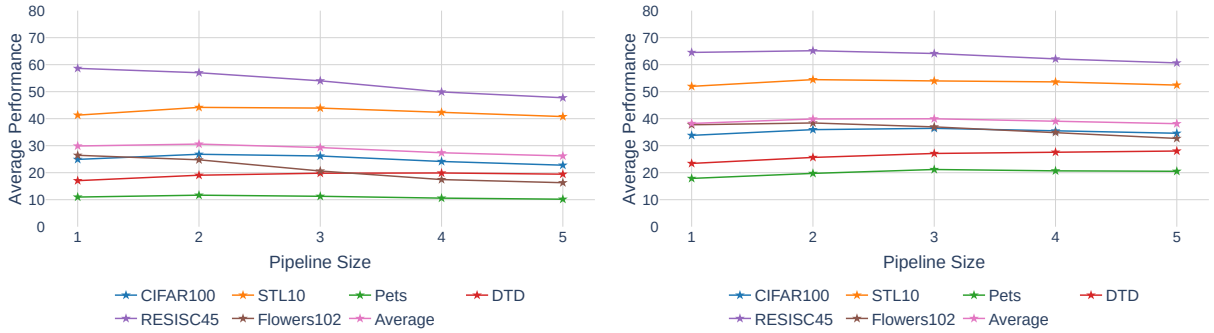
In this experiment, we investigate the impact of the pipeline size K on the performance of our method. The pipeline size determines the number of operations applied to an

image during the transformation process. Since we are exploring the Probabilistic update rule, this pipeline size corresponds to the number of sequential sampling processes performed on a single transformation (refer to Section 4.3.2). Considering the computational requirements of different augmentation operations, we aim to determine the optimal number of operations for achieving reasonable performance in the hope that a smaller pipeline size might yield competitive results, even though this may seem counterintuitive.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.11: Average performance of RU on SimCLR at various pipeline sizes for different downstream tasks.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.12: Average performance of RU (with uniform ranker) on SimCLR at various pipeline sizes for different downstream tasks.

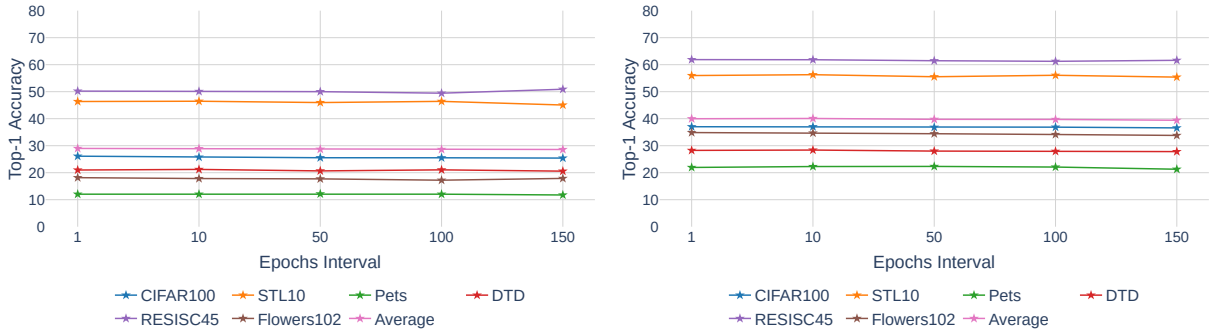
We evaluate our method across various pipeline sizes, specifically ranging from 1 to 5. As illustrated in Figure 6.11, there is a discernible decline in performance with increasing pipeline size across most downstream tasks, a trend particularly evident in the KNN evaluation. These findings suggest that pipeline size exerts a measurable influence on the performance of our method, with less computationally intensive augmentation pipelines being more favorable. To further investigate whether this phenomenon is independent of any particular operation favored by our ranking pipeline, we replace the ranking step with one that returns a uniform probability distribution across all operations in the augmentation set. We observe a similar trend for KNN evaluation in Figure 6.11 as in Figure 6.12, suggesting that the downward trend observed earlier

might be primarily due to factors independent of our ranking mechanism and may instead be a consequence of using a stronger augmentation pipeline as the pipeline size increases.

Effect of the Epochs Interval

Next, we explore how the frequency of the rank-update process affects the performance of our method. In this experiment, we test our method with various epoch intervals, specifically, intervals of 1, 10, 50, 100, and 150 epochs. It is worth noting that in our experiments, we run training for a total of 300 epochs. Therefore, a 1-epoch interval means ranking is performed before each epoch, while a 150-epoch interval implies ranking is conducted twice during the entire pre-training process.

In Figure 6.13, we notice a trend similar to what we observed in Figure 6.10, where there is minimal change in the performance of our method as we vary the epoch interval. This stability in performance regarding the epoch interval parameter for the chosen SSL configuration suggests that our method is robust in this regard. However, it also raises the question of whether the selected ranking mechanism is essential for this specific SSL configuration. Hence, further investigation may be required to determine the relevance of our method and its various components.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.13: Average performance of RU on SimCLR at various pre-training epochs intervals for different downstream tasks.

Effect of the Downstream Tasks during Ranking

In Section 6.2, we used all downstream tasks for the ranking process. This section explores how choosing different tasks for ranking affects the model's performance when evaluated on other datasets. We aim to answer two key questions:

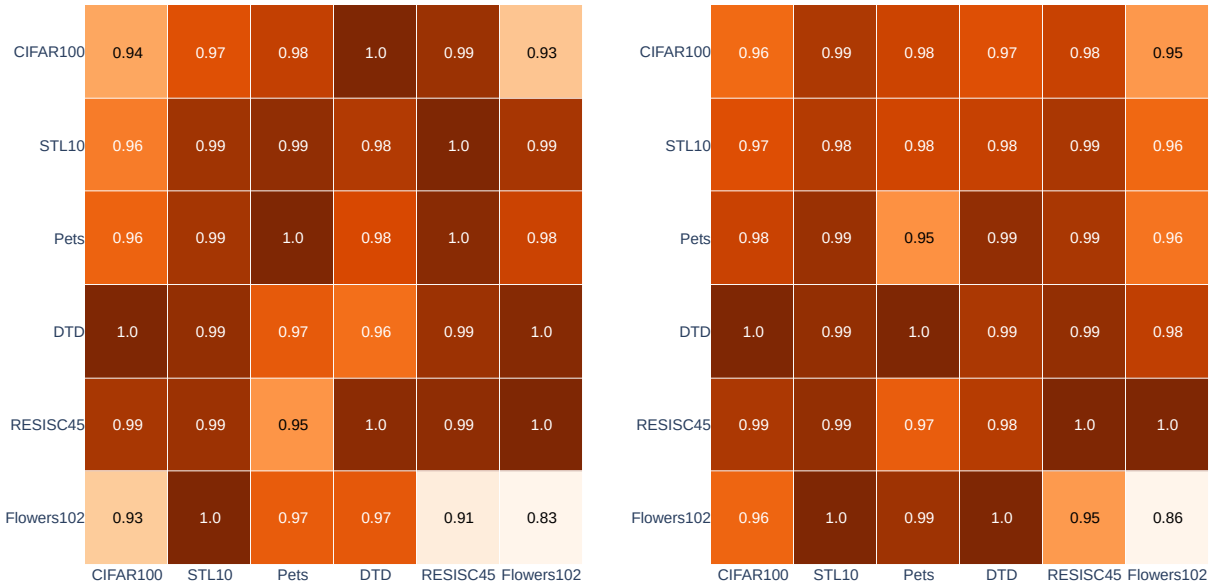
1. **Does evaluating and ranking on the same task lead to the best perfomance for that tasks?** In other words, does a model achieve optimal performance on a specific task when its pre-training augmentation pipeline is constructed exclusively based on ranking from that same task?

2. **How do the downstream tasks used for ranking influence the final rankings of operations?** This question explores whether certain tasks influence the ranking

towards operations that excel on those tasks, possibly neglecting operations that perform better in different tasks.

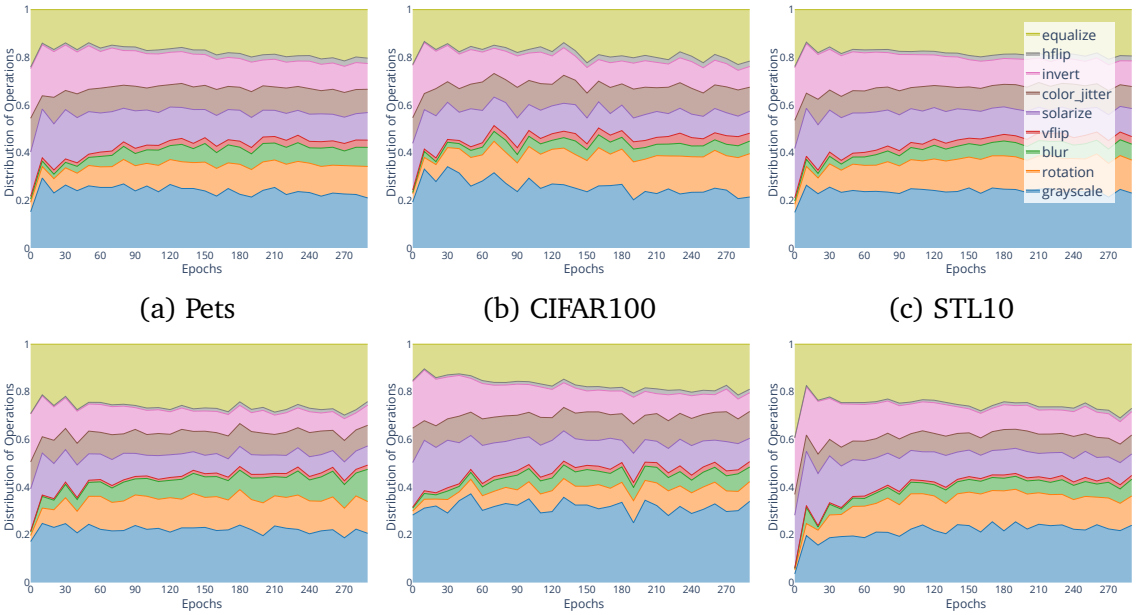
To address these questions, we conduct experiments where a single downstream task is used for ranking, while the model's performance is evaluated on different downstream tasks. The results in Figure 6.14 depict a heatmap showing the average performance for each pairing of ranking and evaluation downstream tasks. The rows of the heatmap represent the downstream task used for ranking, and the columns represent the downstream tasks used to evaluate the model. We normalize the heatmap by dividing each value in a column by the maximum value in that column, highlighting the best-performing ranking task for each evaluation task.

The chosen ranking method may not prioritize the tasks it was designed for, as evidenced by the inconsistent performance in the heatmaps. Figure 6.15 shows similar probability distributions across tasks, indicating the need to update the ranking mechanism. However, these results also elucidate our method's underperformance on *Flowers102* and *RESISC45* compared to other augmentation methods (refer to Section 6.2.2). Essentially, our ranking mechanism tends to enhance performance for some tasks while diminishing it for others because it prioritizes the same set of transforms regardless of the downstream task used for ranking.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.14: A heatmap illustrating the performance of RU with SimCLR for various combinations of ranking and evaluation downstream tasks. The rows correspond to the downstream task used for ranking, while the columns represent the downstream tasks used for evaluating the model.



(a) Pets (b) CIFAR100 (c) STL10

(d) DTD (e) Flowers102 (f) RESISC45

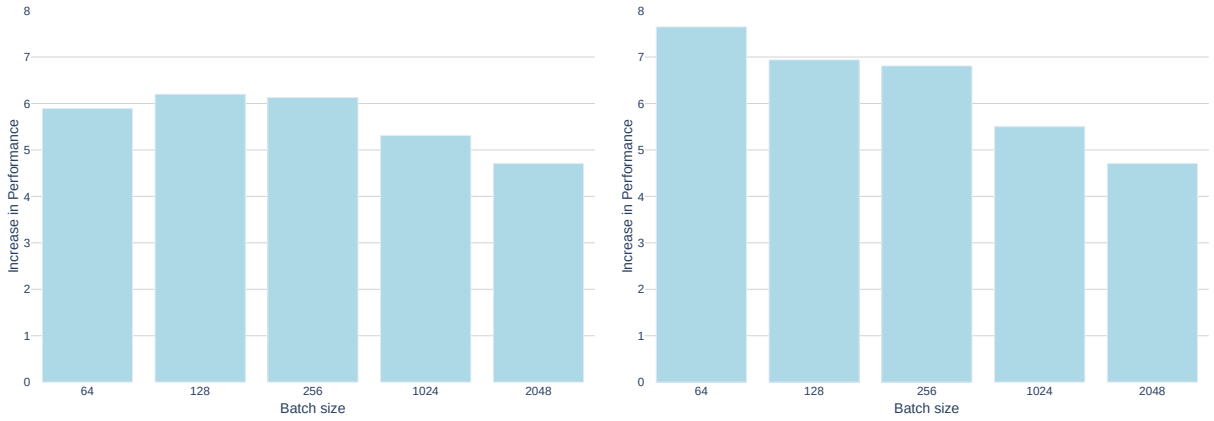
Figure 6.15: Stacked area plot of the evolution probabilities of different operations in the augmentation bag $\mathcal{T}$ when the ranking is based on a single downstream task (each row in Figure 6.14)

6.3.2 Ablations over SSL configurations

After exploring the impacts of different configurations specific to our method, we now shift our attention to more general configurations shared among all contrastive SSL methods. This section aims to examine how these parameters affect the performance improvements over a fixed augmentation pipeline. For these experiments, we will maintain the same Rank-Update (RU) configurations as outlined in Section 6.2. However, we will vary the batch size, the number of pre-training epochs, and the encoder architecture of our pre-training process. This selection is justified by the well-established sensitivity of contrastive SSL models to these hyperparameters [Chen *et al.* 2020b]. It is important to emphasize that our main focus is on the performance difference between our method and a fixed augmentation pipeline. Therefore, we will analyze the variation in average performance rather than the absolute performance on each task, as we did in Section 6.2.

Effect of the Batch Size

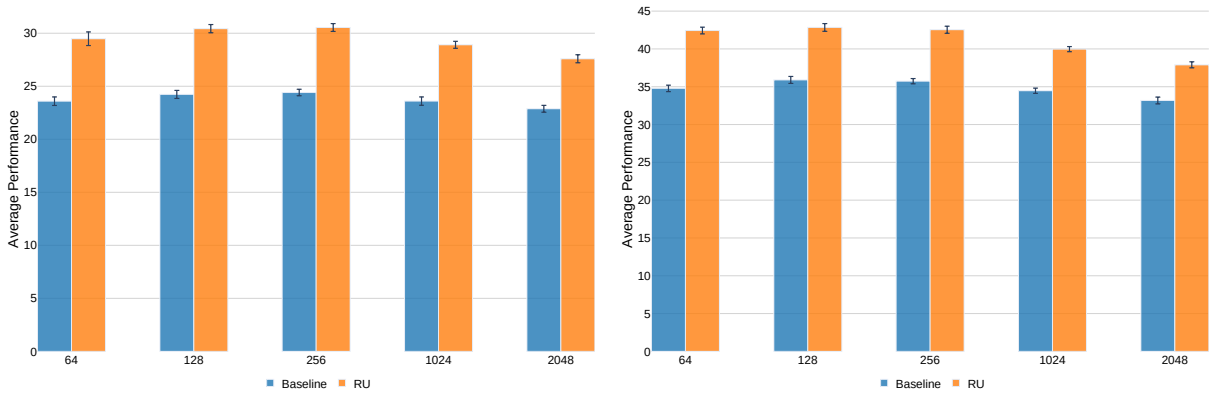
We execute our method with five distinct batch sizes, specifically $[64, 128, 256, 1024, 2048]$. The batch size holds particular significance in contrastive SSL, particularly with techniques that use negative samples repulsion in their contrastive objective function such as SimCLR [Chen *et al.* 2020b].



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.16: Difference in Average Performance of RU over a fixed augmentation pipeline on SimCLR at various batch sizes.

From the results presented in Figure 6.16, it is evident that our method consistently outperforms a fixed augmentation pipeline across all batch sizes. This is indicated by the positive values of the bars, representing an increase in performance over a fixed augmentation pipeline. However, it is noteworthy that the performance gains diminish as the batch size increases during linear evaluation. Upon examining the results in Figure 6.17, we can infer that the performance gains diminish because our method's performance decreases more rapidly than that of a fixed augmentation pipeline as we increase the batch size, suggesting that our method is less robust to larger batch sizes.



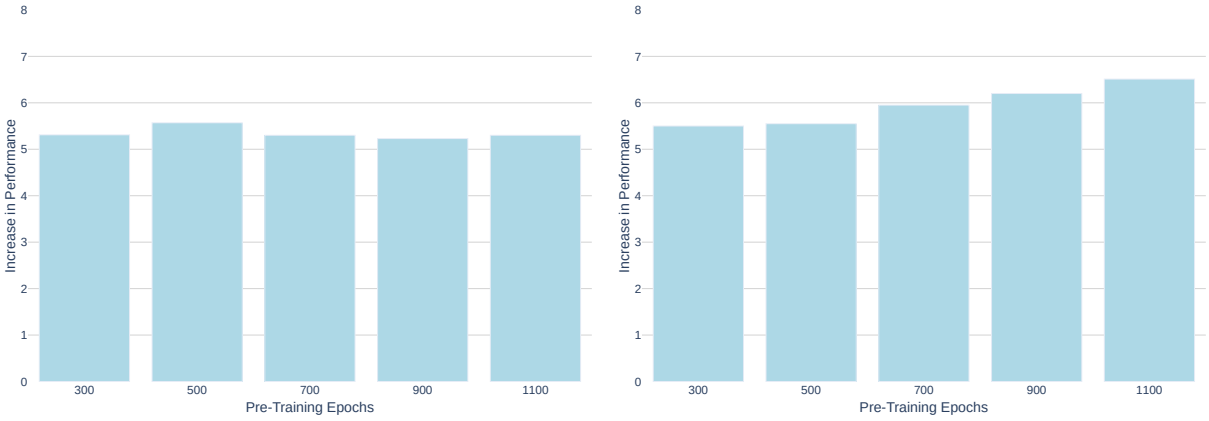
(a) KNN Evaluation (b) Linear Evaluation

Figure 6.17: Average Performance of RU over a fixed augmentation pipeline on SimCLR for different batch sizes.

Effect of the Pre-training Epochs

The number of pre-training epochs is another critical aspect of the SSL pipeline. Literature often suggests using a sufficiently large number of pre-training epochs to improve

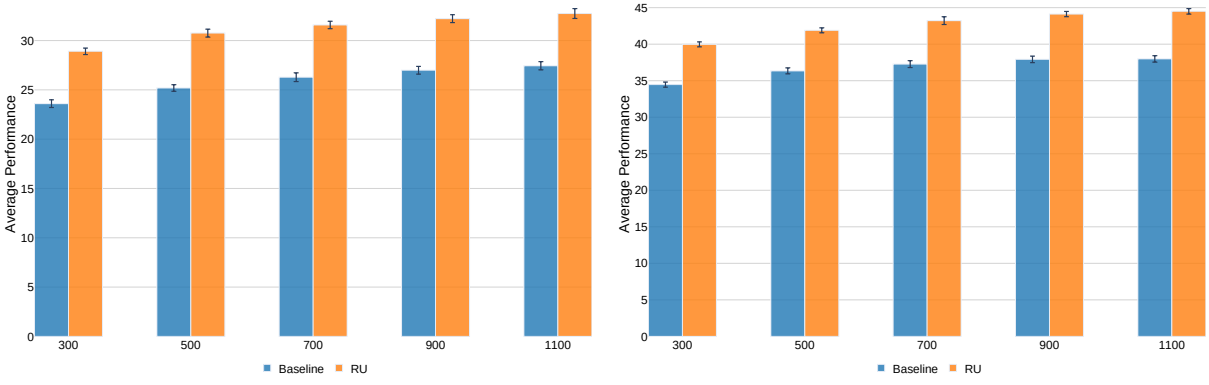
performance, although the extent of performance gains tends to diminish as the number of epochs increases [Chen *et al.* 2020b]. In this experiment, we vary the number of pre-training epochs, specifically testing values of $[300, 500, 700, 900, 1100]$.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.18: Difference in Average Performance of RU over a fixed augmentation pipeline on SimCLR at various pre-training epochs.

While we also observe a consistent performance gain across all pre-training epochs, as shown in Figure 6.18, we notice that the performance gains increase as the number of pre-training epochs increases for linear evaluation. This contrasts the results obtained for KNN evaluation, where we see a relatively stable performance gain across all pre-training epochs. Upon examining the results in Figure 6.19, we observe that the performance of our method increases more rapidly than that of a fixed augmentation pipeline when we perform linear evaluation.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.19: Average Performance of RU over a fixed augmentation pipeline on SimCLR for different pre-training epochs.

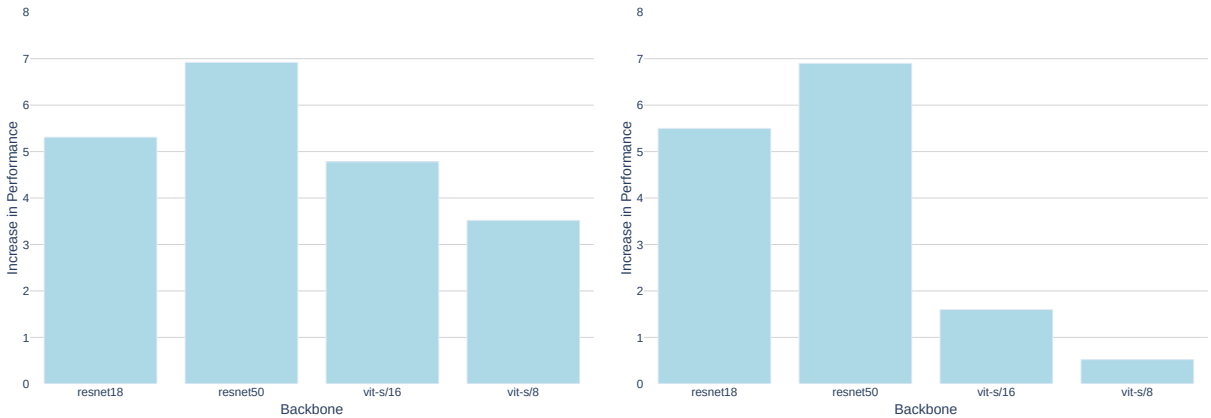
Effect of the Encoder Architecture

Lastly, we vary the encoder architecture used for pre-training, which maps the images to a representation space. The choice of the encoder architecture often determines the ease with which invariances are learned. Some studies even demonstrated the benefits of using non-CNN architectures for different computer vision tasks [Dosovitskiy *et al.* 2020]. The architectures used are outlined in Table 6.6.

Table 6.6: Encoder architectures used during pre-training with SimCLR. Each architecture has two properties attached. **#Heads** which represents the number of heads used for Transformer-based architectures and **#Params** which represents the number of parameters in the encoder.

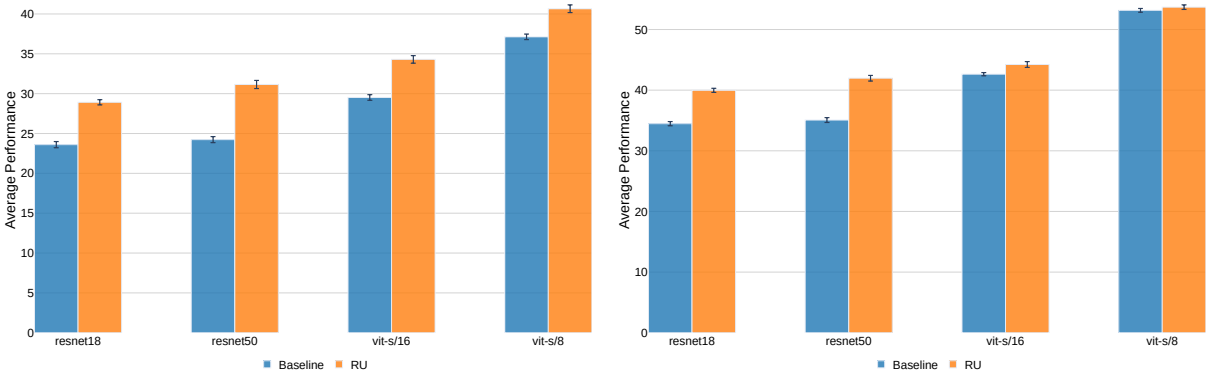
Architecture	**#Heads**	**#Params**
ResNet18 [He *et al.* 2016]	—	11M
ResNet50 [He *et al.* 2016]	—	23M
ViT-S/16 [Dosovitskiy *et al.* 2020]	16	21M
ViT-S/8 [Dosovitskiy *et al.* 2020]	8	21M

While analyzing the results for different encoders in Figure 6.20, we notice that our method offers greater performance gains for CNN-based architectures compared to Transformer-based ones. This contrast becomes even more apparent when considering the performance gains for linear evaluation. We plot the absolute performance for different architectures to understand the cause of this trend.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.20: Difference in Average Performance of RU over a fixed augmentation pipeline on SimCLR for different encoders.



(a) KNN Evaluation (b) Linear Evaluation

Figure 6.21: Average Performance of RU over a fixed augmentation pipeline on SimCLR for different encoders.

As shown in Figure 6.21, the decrease in performance gains for Transformer-based architectures cannot be solely attributed to their inherently lower performance. On the contrary, we observe that the performance of these architectures exceeds that of CNN-based architectures, even surpassing architectures like ResNet50, which has more learnable parameters. This indicates that the performance gains achieved solely by using Transformer-based architectures overshadow the gains produced by our method.

6.4 Summary

In this chapter, we conducted a series of experiments focused on the augmentation method in contrastive SSL. Our goal was to explore data augmentation in contrastive SSL from three different perspectives. We will now summarize the findings for each of them in the following paragraphs.

How the Augmentation Pipeline Influences Performance in contrastive SSL

Our analysis of the SSL algorithm's performance with different augmentation pipelines has revealed a strong dependence on the chosen augmentation pipeline. This dependence becomes more pronounced with the increase in pipeline size, transitioning from a single stage to a 2-stage pipeline as seen in Section 6.1.2. Overall, these results effectively highlight the necessity for an enhanced augmentation method capable of adapting to various downstream tasks.

Furthermore, in Section 6.1.2, we observed that certain tasks exhibited similar performance trends across all conducted experiments within that section. This observation suggests that these tasks may share specific characteristics that predispose them to favor the same set of operations, despite belonging to unrelated domains. For instance, both *RESISC45* (comprising satellite images) and *Flowers102* (featuring 102 classes of flowers) demonstrated similar performance trends despite belonging to different domains.

How our Augmentation Framework Compares to other Methods

Our method has demonstrated superior performance compared to other parameter-free augmentation methods, such as RandAugment and TrivialAugment, as well as a fixed augmentation pipeline. However, this success was achieved by employing the Probabilistic update rule, which relies on stochastic sampling of augmentation operations based on probabilities obtained from the chosen ranking mechanism. While this method has proven to be effective, its success is likely more attributed to the diversity introduced by the stochastic sampling process than to the ranking mechanism itself, as observed from the results in Section 6.2.1.

While our method outperformed others, its performance was comparatively low for specific downstream tasks, particularly Flowers102 and RESISC45. Our method consistently underperformed on these tasks compared to others. A closer look at the results in Section 6.1 reveals that Flowers102 and RESISC45 share a similar affinity for rotation as augmentation operations. Unfortunately, our method favored color-distorting operations such as *Equalize* or *Grayscale*. This suggests that our ranking mechanism might struggle to capture this affinity due to the influence of other tasks favoring different operations during the ranking process.

How our Method is Impacted by Different Training Configurations

Our method has exhibited little sensitivity to its various configurations, except for the pipeline size and the downstream tasks used for ranking. We observed a slight decrease in performance as the pipeline size increased. Additionally, the ranking mechanism could not capture the relevant probability distribution that favored a particular downstream task the most when that downstream task was exclusively used during ranking (see Figure 6.14). This might indicate the need for a more accurate ranking mechanism that can capture the necessary information required for ranking.

Furthermore, our method maintains its performance gains over a fixed augmentation pipeline, especially for KNN evaluation. The performance gains increase as the number of pre-training epochs increases (see Section 6.3.2), while they decrease as the batch size increases (see Section 6.3.2). Additionally, we discovered that our method provides greater benefits to CNN-based architectures compared to Transformer-based ones. This is primarily because Transformer-based architectures already perform well, even with a fixed augmentation pipeline, thereby reducing the potential gains offered by our method.

Chapter 7

Conclusion & Future Work

The significance of contrastive SSL methods has been well-established through their applications across various vision tasks [Oquab *et al.* 2024; Chen *et al.* 2020b; Zbontar *et al.* 2021; Chen and He 2021]. However, the challenge of learning a generalized representation space that can be used for a range of downstream tasks remains a significant hurdle in the field. This challenge is further compounded by the reliance of these methods on the augmentation pipeline utilized during pre-training. In practice, certain augmentation pipelines may introduce invariances in the representation space that are suitable for some tasks but not for others [Wagner *et al.* 2022; Balestriero *et al.* 2023]. It is this observation that served as the motivation for our work.

As a solution to this problem, we proposed the Rank-Update data augmentation framework. This framework enables the creation of a dynamic augmentation pipeline by ranking operations based on their relevance to downstream tasks. Our method does not rely on a set of parameters to be learned and optimized, making it easy to integrate into any contrastive SSL methods without imposing computational overhead or requiring modifications to other components of the pre-training process aside from the augmentation process.

We evaluated our method across various downstream tasks and showcased its superiority over a fixed augmentation pipeline designed for ImageNet-like datasets (STL10, CIFAR100). Furthermore, we compared our method to other parameter-free methods utilizing a similar stochastic sampling process and demonstrated its superiority in several of the downstream tasks used for evaluation.

Unfortunately, the results in Section 6.3.1, where we ranked based on an individual downstream task, did not demonstrate that ranking solely based on a specific downstream task resulted in optimal performance when evaluating our model on the same downstream task. Additionally, it revealed that our method favored the same set of operations regardless of the downstream task used during the ranking process. This suggests that while we observed performance improvements on some downstream tasks, the ranking mechanism based on KL-Divergence needs to be more effective in identifying operations relevant to specific downstream tasks. Consequently, most of the benefits of this technique arise from the diversity introduced by the stochastic sampling

of operations in our Probabilistic update rule. This underscores the need to design better ranking mechanisms that accurately capture the relevance of an augmentation operation to a downstream task.

In the future, an intriguing avenue for research would involve exploring the integration of bandit theory into our ranking mechanism [Sutton and Barto 2018]. In this scenario, the **arms** of the bandit algorithms could be the different augmentation operations available to the algorithm, and the **reward** could be the performance gains on different downstream tasks. Consequently, the algorithm would prioritize operations that maximize performance improvements across various downstream tasks. To implement this technique during pre-training, we would need to devise a reward mechanism that can be computed within the SSL training framework. One potential approach could involve measuring the KNN evaluation on various downstream tasks using the learned representation space of the currently trained encoder, as it requires fewer computational resources compared to linear evaluation or fine-tuning.

In conclusion, we introduced a novel augmentation framework capable of constructing a dynamic augmentation pipeline based on the relevance of operations to a set of downstream tasks. However, we encountered challenges in developing a ranking mechanism that effectively captures this relevance. Nevertheless, we demonstrated that our method can outperform other parameter-free augmentation methods and a fixed augmentation pipeline on specific downstream tasks. The framework is appealing due to its simplicity and ability to seamlessly integrate into any contrastive SSL methods without requiring modifications to the original pre-training process. We hope this work inspires future research in the field, encouraging further investigation into data augmentation in contrastive SSL.

Bibliography

[Addepalli *et al.* 2022] Sravanti Addepalli, K. Bhogale, P. Dey, and R. Venkatesh Babu. Towards efficient and effective self-supervised learning of visual representations. *European Conference on Computer Vision*, 2022.

[Agrawal *et al.* 2022] Kumar Krishna Agrawal, Arnab Kumar Mondal, Arna Ghosh, and Blake Aaron Richards. α-ReQ : Assessing representation quality in self-supervised learning by measuring eigenspectrum decay. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[Alosaimi *et al.* 2023] Najd Alosaimi, Haikel Alhichri, Yakoub Bazi, Belgacem Ben Youssef, and Naif Alajlan. Self-supervised learning for remote sensing scene classification under the few-shot scenario. *Scientific Reports*, 13(1):433, 2023.

[Assran *et al.* 2023] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15619–15629, June 2023.

[Baldi 2011] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, page 37–50. JMLR.org, 2011.

[Balestriero *et al.* 2023] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning. *arXiv preprint arXiv: 2304.12210*, 2023.

[Bandara *et al.* 2023] Wele Gedara Chaminda Bandara, Celso M. De Melo, and Vishal M. Patel. Guarding barlow twins against overfitting with mixed samples. *arXiv preprint arXiv: 2312.02151*, 2023.

[Bao *et al.* 2021] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *International Conference on Learning Representations*, 2021.

[Bardes *et al.* 2021] Adrien Bardes, J. Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *International Conference on Learning Representations*, 2021.

[Bengio *et al.* 2013] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35:1798–1828, 2013.

[Bengio 2012] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 17–36, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[Bordes *et al.* 2023] Florian Bordes, Randall Balestriero, and Pascal Vincent. Towards democratizing joint-embedding self-supervised learning. *arXiv preprint arXiv: 2303.01986*, 2023.

[Caron *et al.* 2018] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. *European Conference on Computer Vision*, 2018.

[Caron *et al.* 2020] Mathilde Caron, Ishan Misra, J. Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Neural Information Processing Systems*, 2020.

[Caron *et al.* 2021] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9650–9660, October 2021.

[Chen and He 2021] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15750–15758, June 2021.

[Chen *et al.* 2020a] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1691–1703. PMLR, 13–18 Jul 2020.

[Chen *et al.* 2020b] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *International Conference on Machine Learning*, 2020.

[Chen *et al.* 2020c] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv: 2003.04297*, 2020.

[Cheng *et al.* 2017] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, Oct 2017.

[Chopra *et al.* 2005] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, pages 539–546. IEEE Computer Society, 2005.

[Chuang *et al.* 2020] Ching-Yao Chuang, Joshua Robinson, Yen-Chen Lin, Antonio Torralba, and Stefanie Jegelka. Debiased contrastive learning. *Advances in neural information processing systems*, 33:8765–8775, 2020.

[Cimpoi *et al.* 2014] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[Coates *et al.* 2011] Adam Coates, Honglak Lee, and Andrew Y. Ng. An analysis of single layer networks in unsupervised feature learning. In *AISTATS*, 2011.

[Collobert *et al.* 2011] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(76):2493–2537, 2011.

[Conneau *et al.* 2019] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *Annual Meeting of the Association for Computational Linguistics*, 2019.

[Csiszar 1975] I. Csiszar. I-Divergence Geometry of Probability Distributions and Minimization Problems. *The Annals of Probability*, 3(1):146 – 158, 1975.

[Cubuk *et al.* 2019] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[Cubuk *et al.* 2020] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 3008–3017. Computer Vision Foundation / IEEE, 2020.

[Dangovski *et al.* 2022] Rumen Dangovski, Li Jing, Charlotte Loh, Seungwook Han, Akash Srivastava, Brian Cheung, Pulkit Agrawal, and Marin Soljacic. Equivariant self-supervised learning: Encouraging equivariance in representations. In *International Conference on Learning Representations*, 2022.

[Devlin *et al.* 2019] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language

understanding. *North American Chapter of the Association for Computational Linguistics*, 2019.

[Doersch *et al.* 2015] Carl Doersch, A. Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. *IEEE International Conference on Computer Vision*, 2015.

[Dosovitskiy *et al.* 2020] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, S. Gelly, Jakob Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2020.

[Erhan *et al.* 2010] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, mar 2010.

[Ericsson *et al.* 2022] Linus Ericsson, Henry Gouk, and Timothy Hospedales. Why do self-supervised models transfer? on the impact of invariance on downstream tasks. In *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022.

[Frosst *et al.* 2019] Nicholas Frosst, Nicolas Papernot, and Geoffrey E. Hinton. Analyzing and improving representations with the soft nearest neighbor loss. *International Conference on Machine Learning*, 2019.

[Garrido *et al.* 2022] Q. Garrido, Randall Balestriero, Laurent Najman, and Yann LeCun. Rankme: Assessing the downstream performance of pretrained self-supervised representations by their rank. *International Conference on Machine Learning*, 2022.

[Gidaris *et al.* 2018] Spyros Gidaris, Praveer Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations. *International Conference on Learning Representations*, 2018.

[Grill *et al.* 2020] Jean-Bastien Grill, Florian Strub, Florent Altch'e, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, B. '. Pires, Z. Guo, M. G. Azar, Bilal Piot, K. Kavukcuoglu, R. Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *Neural Information Processing Systems*, 2020.

[Group 2017] Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[Gudovskiy *et al.* 2021] Denis A. Gudovskiy, L. Rigazio, Shun Ishizaka, K. Kozuka, and Sotaro Tsukizawa. Autodo: Robust autoaugment for biased data with label noise via scalable probabilistic implicit differentiation. *Computer Vision and Pattern Recognition*, 2021.

[Gutmann and Hyvärinen 2010] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical

models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[Hataya *et al.* 2019] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. *European Conference on Computer Vision*, 2019.

[Hataya *et al.* 2022] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Meta approach to data augmentation optimization. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2574–2583, January 2022.

[He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[He *et al.* 2020] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *CVPR*, 2020.

[He *et al.* 2021] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll'ar, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *Computer Vision and Pattern Recognition*, 2021.

[Jaiswal *et al.* 2020] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self- supervised learning. *Technologies*, 2020.

[Jing *et al.* 2022] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. In *International Conference on Learning Representations*, 2022.

[Kingma and Ba 2014] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[Kingma and Welling 2013] Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations,* 2013.

[Krizhevsky and Hinton 2009] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Technical report, University of Toronto, 2009.

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[Le and Yang 2015] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. 2015.

[Le *et al.* 2011] Quoc V. Le, Marc'Aurelio Ranzato, R. Monga, M. Devin, G. Corrado, Kai Chen, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2011.

[Li *et al.* 2020] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M. Robertson, and Yongxin Yang. Dada: Differentiable automatic data augmentation. *ECCV*, 2020.

[Libovický *et al.* 2020] Jindřich Libovický, Rudolf Rosa, and Alexander Fraser. On the language neutrality of pre-trained multilingual representations. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1663–1674, Online, November 2020. Association for Computational Linguistics.

[Lim *et al.* 2019] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. *Advances in Neural Information Processing Systems*, 32, 2019.

[Liu *et al.* 2021] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering*, 35(1):857–876, 2021.

[Mann and Whitney 1947] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50 – 60, 1947.

[Mehta *et al.* 2022] Sachin Mehta, Saeid Naderiparizi, Fartash Faghri, Maxwell Horton, Lailin Chen, Ali Farhadi, Oncel Tuzel, and Mohammad Rastegari. Rangeaugment: Efficient online augmentation with range learning. *arXiv preprint arXiv: 2212.10553*, 2022.

[Mikolov *et al.* 2013] Tomas Mikolov, Kai Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *International Conference on Learning Representations*, 2013.

[Misra and Maaten 2019] Ishan Misra and L. Maaten. Self-supervised learning of pretext-invariant representations. *Computer Vision and Pattern Recognition*, 2019.

[Miyai *et al.* 2022] Atsuyuki Miyai, Qing Yu, Daiki Ikami, Go Irie, and K. Aizawa. Rethinking rotation in self-supervised contrastive learning: Adaptive positive or negative data augmentation. *IEEE Workshop/Winter Conference on Applications of Computer Vision*, 2022.

[Mnih *et al.* 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv: 1312.5602*, 2013.

[Müller and Hutter 2021] Samuel G. Müller and F. Hutter. Trivialaugment: Tuning-free yet state-of-the-art data augmentation. *IEEE International Conference on Computer Vision*, 2021.

[Ni *et al.* 2022] Renkun Ni, Manli Shu, Hossein Souri, Micah Goldblum, and Tom Goldstein. The close relationship between contrastive learning and meta-learning. In *International Conference on Learning Representations*, 2022.

[Nilsback and Zisserman 2008] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[Noroozi and Favaro 2016] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. *European Conference on Computer Vision*, 2016.

[Oquab *et al.* 2024] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

[Parkhi *et al.* 2012] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[Patacchiola and Storkey 2020] Massimiliano Patacchiola and Amos J Storkey. Self-supervised relational reasoning for representation learning. *Advances in Neural Information Processing Systems*, 33:4003–4014, 2020.

[Pathak *et al.* 2016] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. *Computer Vision and Pattern Recognition*, 2016.

[Razavi *et al.* 2019] Ali Razavi, Aäron van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Neural Information Processing Systems*, 2019.

[Reed *et al.* 2021] Colorado J Reed, Sean Metzger, Aravind Srinivas, Trevor Darrell, and Kurt Keutzer. Selfaugment: Automatic augmentation policies for self-supervised learning. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2673–2682, 2021.

[Russakovsky *et al.* 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015.

[Sanh *et al.* 2019] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *NEURIPS*, 2019.

[Schroff *et al.* 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *Computer Vision and Pattern Recognition*, 2015.

[Seltzer and Zhang 2009] Michael L. Seltzer and Lei Zhang. The data deluge: Challenges and opportunities of unlimited data in statistical signal processing. *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3701–3704, 2009.

[Seo *et al.* 2016] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *International Conference on Learning Representations*, 2016.

[Sidheekh *et al.* 2022] Sahil Sidheekh, Chris B. Dock, Tushar Jain, R. Balan, and M. Singh. Vq-flows: Vector quantized local normalizing flows. *Conference on Uncertainty in Artificial Intelligence*, 2022.

[Sohn 2016] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[Sun *et al.* 2017] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *IEEE International Conference on Computer Vision*, 2017.

[Sutton and Barto 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[Tian *et al.* 2020] Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6827–6839. Curran Associates, Inc., 2020.

[Torpey and Klein 2021] David Torpey and Richard Klein. *Explicit homography estimation improves contrastive self-supervised learning*, 2021.

[van den Oord *et al.* 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv: 1807.03748*, 2018.

[Vaswani *et al.* 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Vincent *et al.* 2008] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery.

[Vincent *et al.* 2010] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(110):3371–3408, 2010.

[Wagner *et al.* 2022] Diane Wagner, Fabio Ferreira, Danny Stoll, Robin Tibor Schirrmeister, Samuel Müller, and Frank Hutter. On the importance of hyperparameters and data augmentation for self-supervised learning. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022.

[Wang and Qi 2021] Xiao Wang and Guo-Jun Qi. Contrastive learning with stronger augmentations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[Wang *et al.* 2021] Yifei Wang, Zhengyang Geng, Feng Jiang, Chuming Li, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Residual relaxation for multi-view representation learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12104–12115. Curran Associates, Inc., 2021.

[Wu *et al.* 2018] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[Zbontar *et al.* 2021] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. *International Conference on Machine Learning*, 2021.

[Zhang and Ma 2022] Junbo Zhang and Kaisheng Ma. Rethinking the augmentation module in contrastive learning: Learning hierarchical augmentation invariance with expanded views. *Computer Vision and Pattern Recognition*, 2022.

[Zhang *et al.* 2015] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657, 2015.

[Zheng *et al.* 2022] Yu Zheng, Z. Zhang, Shen Yan, and Mi Zhang. Deep autoaugment. *International Conference on Learning Representations*, 2022.

[Zhou *et al.* 2022] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Yuille, and Tao Kong. Image BERT pre-training with online tokenizer. In *International Conference on Learning Representations*, 2022.

Appendix A

Appendix

A.1 Datasets Details

This section provides a detailed overview of all the datasets used in our experiments. Some examples of images from each dataset are shown in Figure A.1.

TinyImageNet

TinyImageNet is a dataset containing 200 image classes [Le and Yang 2015], with each class having 500 training images, 50 validation images, and 50 test images. Each image has a resolution of 64x64 pixels. It is a subset of the ImageNet dataset [Russakovsky *et al.* 2015] and is commonly used for pre-training deep learning models due to its smaller size while still providing a diverse set of images across multiple classes [Zhang and Ma 2022].

Pets

This dataset comprises approximately 7,000 images featuring 37 different breeds of cats and dogs. Sourced from the Oxford-IIIT Pet Dataset [Parkhi *et al.* 2012] is a valuable resource for pet image classification tasks. While the exact image sizes may vary, but it is popularly used to evaluate the transfer capabilities of SSL models [Chen *et al.* 2020b; Oquab *et al.* 2024].

Describable Textures Dataset (DTD)

With a collection of 5,640 images, DTD is designed for texture recognition applications [Cimpoi *et al.* 2014]. It encompasses 47 texture categories, including fabric, foliage, and glass. The dataset is valuable for understanding and classifying a diverse range of image textures. We selected this dataset because it deviated from natural images, contributing to diversifying our set of evaluation tasks.

Flowers102

The Flowers102 dataset, derived from The Oxford Flowers 102 Dataset [Nilsback and Zisserman 2008], consists of 8,189 images featuring 102 different types of flowers. This dataset is a benchmark for fine-grained flower classification tasks, offering various floral images for research and evaluation purposes.

RESISC45

The NWPU-RESISC45 dataset [Cheng *et al.* 2017], encompassing 31,500 images capturing scenes observed in remote sensing images, offers a diverse array of 45 classes, including beach, desert, and farmland. This dataset is pivotal in advancing remote sensing image analysis and scene classification research. It proves particularly valuable for assessing the transfer capabilities of SSL models on satellite imagery, especially when pre-trained on natural images [Alosaimi *et al.* 2023].

STL10

STL10 [Coates *et al.* 2011], an image recognition dataset inspired by CIFAR-10, comprises 113,000 unlabeled and 5,000 labeled images, encompassing ten classes such as airplane, bird, and car. Renowned for its diverse collection of real-world images, STL10 is versatile for many vision tasks. Its similarity to TinyImageNet, our pre-training dataset, positions it as a valuable resource for assessing the in-domain transfer capabilities of SSL models [Patacchiola and Storkey 2020; Bandara *et al.* 2023].

CIFAR100

The CIFAR100 dataset comprises 60,000 images categorized into 100 classes [Krizhevsky and Hinton 2009]. Much like STL10, CIFAR100 is widely recognized for assessing in-domain transfer capabilities of SSL models [Bandara *et al.* 2023; Chen *et al.* 2020b], introducing additional complexity with its fine-grained annotations of 100 classes.

Dataset	Num Classes	Num Samples	Image Size
Pets	37	$\approx$ 7,000	Varies
DTD	47	5,640	Varies
Flowers102	102	8,189	Varies
RESISC45	45	31,500	Varies
STL10	10	118,000	92x92
CIFAR100	100	60,000	32x32
TinyImageNet	200	100,000	64x64

Table A.1: Information about different datasets used in our experiments.



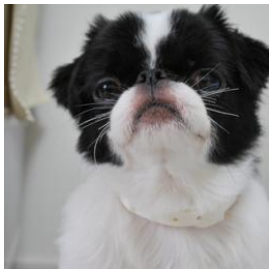
(a) CIFAR100



(b) DTD



(c) Flowers102



(d) Pets



(e) RESISC45



(f) STL-10



(g) Tiny ImageNet

Figure A.1: Example images from various datasets. Some images may appear blurry as they have been resized to match the resolution of other images. The original sizes of images from each dataset are provided in Table A.1.

A.2 Implementation Details & Hyperparameters

In this section, we provide details about implementing experimentation pipeline. We also provide details about the hyperparameters used in our experiments.

A.2.1 Pre-training

During pre-training, we train the `encoder` for 300 epochs, following the pre-training methods highlighted in Section 5.2, using the Adam optimizer [Kingma and Ba 2014]. We utilize a batch size of 1024, which consists of images from the TinyImageNet dataset resized to 32x32 pixels. We provide more details in Table A.2.

Parameter	Value
`epochs`	300
`batch_size`	1024
`image_size`	32x32
`encoder`	resnet18 [He *et al.* 2016]
`projector`	2-Layer MLP (512, ReLU, 128)
`optimizer`	Adam(learning_rate=1e-4, weight_decay=1e-6)

Table A.2: Hyperparameters for Pre-training

A.2.2 Linear Evaluation

After pre-training the `encoder`, we extract the learned representations and use them to train a separate linear classifier for each downstream task using the hyperparameters highlighted in Table A.3.

Parameter	Value
`epochs`	300
`batch_size`	1024
`image_size`	32x32
`encoder`	resnet18 [He *et al.* 2016] (pre-trained)
`optimizer`	Adam(learning_rate=1e-3, weight_decay=1e-6)

Table A.3: Hyperparameters for Linear Evaluation

Please note that we freeze the weights of the `encoder` obtained during the pre-training stage and only train the linear classifier. It is also important to emphasize that we maintain the same number of `epochs` for every downstream task, except for Flowers102, where we extend the training to **800** epochs. This decision stems from previous hyperparameter tuning, which revealed that the model had not sufficiently converged after 600 epochs in the case of Flowers102.

A.2.3 Data Agumentation

Agumentation Bag for Ranking

We use the following set of augmentation operations to construct the augmentation bag for ranking:

Transform	`TorchVision` [1] Method
hflip	`RandomHorizontalFlip(p=0.5)`
vflip	`RandomVerticalFlip(p=0.5)`
blur	`GaussianBlur(input_size=input_size, p=0.5)`

color_jitter	`RandomApply([ColorJitter(0.8 * color_jitter_strength,`
	`0.8 * color_jitter_strength,`
	`0.8 * color_jitter_strength,`
	`0.2 * color_jitter_strength)], p=0.8)`

grayscale	`RandomGrayscale(p=0.5)`
rotation	`RandomRotation((-127, 127))`
invert	`RandomInvert(p=0.6)`
equalize	`RandomEqualize()`
solarize	`RandomSolarize(threshold=0.5078125, p=0.5)`