Improving Deep Learning Models in Weakly Supervised Learning Paradigms

Fariborz Taherkhani
West Virginia University, ft0009@mix.wvu.edu

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Part of the Engineering Commons

Recommended Citation
https://researchrepository.wvu.edu/etd/8221

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.
Improving Deep Learning Models in Weakly Supervised Learning Paradigms

Fariborz Taherkhani

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

Nasser M. Nasrabadi, Ph.D., Chair
Jeremy Dawson, Ph.D.
Matthew C. Valenti, Ph.D.
Xin Li, Ph.D.
Omid Dehzangi, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2021

Keywords: Semi-Supervised Learning, Deep Learning, Matrix Completion, Sparse Subspace, Low Rank Subspace, Optimal Transport, Self-Supervised Learning

Copyright 2021 Fariborz Taherkhani
Abstract

Improving Deep Learning Models in Weakly Supervised Learning Paradigms

Fariborz Taherkhani

Convolutional Neural Networks (CNNs) have provided promising achievements for image classification problems. However, training a CNN model relies on a large number of labeled data. Considering the vast amount of unlabeled data available on the web, it is important to make use of these data in conjunction with a small set of labeled data to train a deep learning model. In this thesis, we aim to develop some methods that can make use of unlabeled data along with partially labeled data to obtain a better generalization performance for CNN models. Moreover, it is also often easier to collect labels that capture only part of the information about the true label of interest. A particularly pertinent example is semantic labels obtained from hashtags attached to images. Such tags are generally easy to gather in large quantities, but tend to only capture certain aspects of the image that the person tagging them focused on. For example, objects are usually organized in a hierarchical structure in which each coarse category (e.g., big cat) corresponds to a super-class of several fine categories (e.g., cheetah, leopard). The objects grouped within the same coarse category, but in different fine categories, usually share a set of global features; however, these objects have distinctive local properties that characterize them at a fine level. Therefore we can tackle the challenge of fine image classification in a weakly supervised fashion, whereby a subset of images is tagged by fine labels (i.e., fine images), while the remaining are tagged by coarse labels (i.e., coarse images).
Contents

List of Figures vi
List of Tables viii

1 Introduction 1
  1.1 Problem and Motivation ........................................ 1
  1.2 Outline and Contributions ...................................... 2

2 Semi-Supervised Learning Approaches in Deep Learning, and Backgrounds 5
  2.1 Introduction ...................................................... 5
  2.2 Consistency Regularization ...................................... 6
  2.3 Entropy Minimization ........................................... 7
  2.4 Pseudo Labeling .................................................. 7
  2.5 Deep Semi-Supervised Learning ................................ 8
  2.6 Graph-Based Semi-Supervised Learning ......................... 9
  2.7 Hierarchical Structure for Visual Recognition ............... 9
  2.8 Preliminaries, and Background ................................ 10
    2.8.1 Matrix Completion Based on Rank Minimization ........ 10
    2.8.2 Low Rank and Sparse Representation ..................... 11
    2.8.3 Low Rank and Sparse Self-Expressive .................... 11

3 Matrix Completion for Graph-Based Deep Semi-Supervised Learning 14
  3.1 Introduction ...................................................... 14
  3.2 Similarity Graph Construction .................................. 16
    3.2.1 Predicting Missing Labels via Matrix Completion ........ 17
  3.3 Supervised Task Regularization via the Constructed Graph .... 19
    3.3.1 Deep Semi Supervised Loss Function ....................... 20
  3.4 Experiments ...................................................... 21

4 A Weakly Supervised Fine Label Classifier Enhanced by Coarse Supervision 29
  4.1 Introduction ...................................................... 30
  4.2 Supervised Self-Expressive Layers ............................ 32
    4.2.1 Supervised Low Rank Self-Expressive Layer .............. 32
    4.2.2 Supervised Sparse Self-Expressive Layer ................ 34
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3 Fine-Coarse Label Classifier Framework</td>
<td>35</td>
</tr>
<tr>
<td>4.4 Experiments and Implementation Details</td>
<td>36</td>
</tr>
<tr>
<td>4.4.1 CNN Architecture</td>
<td>36</td>
</tr>
<tr>
<td>4.4.2 Training the Framework</td>
<td>36</td>
</tr>
<tr>
<td>4.4.3 Dataset and Experimental Setup</td>
<td>38</td>
</tr>
<tr>
<td>4.4.4 Hyper-parameters Tuning</td>
<td>38</td>
</tr>
<tr>
<td>4.4.5 Evaluating the Model in Different Cases</td>
<td>39</td>
</tr>
<tr>
<td>4.4.6 Impact of Fine and Coarse Samples in Training</td>
<td>41</td>
</tr>
<tr>
<td>4.4.7 Comparison</td>
<td>42</td>
</tr>
<tr>
<td>4.4.8 Further Analysis</td>
<td>43</td>
</tr>
<tr>
<td>5 Hierarchical Optimal Transport for Semi-Supervised Learning</td>
<td>45</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>45</td>
</tr>
<tr>
<td>5.2 Preliminaries</td>
<td>47</td>
</tr>
<tr>
<td>5.2.1 Discrete OT and Dual Form</td>
<td>47</td>
</tr>
<tr>
<td>5.2.2 Hierarchical OT</td>
<td>49</td>
</tr>
<tr>
<td>5.2.3 Wasserstein Barycenters</td>
<td>50</td>
</tr>
<tr>
<td>5.3 Method</td>
<td>50</td>
</tr>
<tr>
<td>5.3.1 Finding Unlabeled Measures via Wasserstein Metric</td>
<td>51</td>
</tr>
<tr>
<td>5.3.2 Mapping Measures via Hierarchical OT for Pseudo-Labeling</td>
<td>53</td>
</tr>
<tr>
<td>5.3.3 Training CNN in SSL Fashion</td>
<td>53</td>
</tr>
<tr>
<td>5.4 Experiments and Setup</td>
<td>55</td>
</tr>
<tr>
<td>5.4.1 Fully Supervised and Deep SSL Methods</td>
<td>57</td>
</tr>
<tr>
<td>5.4.2 Soft-Pseudo-Labels based on Hierarchical OT</td>
<td>57</td>
</tr>
<tr>
<td>5.4.3 Contribution of Hierarchical Optimal Transport to SSL</td>
<td>58</td>
</tr>
<tr>
<td>5.4.4 Clustering Resolution</td>
<td>60</td>
</tr>
<tr>
<td>5.4.5 Varying Labeled Data</td>
<td>61</td>
</tr>
<tr>
<td>5.5 Conclusion</td>
<td>61</td>
</tr>
<tr>
<td>6 Self-Supervised Learning and Optimal Transport for Semi-Supervised Learning</td>
<td>62</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>63</td>
</tr>
<tr>
<td>6.2 Related Work</td>
<td>64</td>
</tr>
<tr>
<td>6.2.1 Semi Supervised Learning for Deep Models</td>
<td>64</td>
</tr>
<tr>
<td>6.2.2 Self-Supervised Learning</td>
<td>66</td>
</tr>
<tr>
<td>6.3 Wasserstein Distance</td>
<td>66</td>
</tr>
<tr>
<td>6.4 Wasserstein Barycenter</td>
<td>67</td>
</tr>
<tr>
<td>6.5 Proposed Method</td>
<td>68</td>
</tr>
<tr>
<td>6.5.1 Self-Supervised Learning and Clustering via Wasserstein Barycenter</td>
<td>69</td>
</tr>
<tr>
<td>6.5.2 Matching Clusters to Classes via WGAN</td>
<td>70</td>
</tr>
<tr>
<td>6.5.3 Total Loss for Training the CNN</td>
<td>72</td>
</tr>
<tr>
<td>6.6 Experiments</td>
<td>73</td>
</tr>
<tr>
<td>6.6.1 Comparison</td>
<td>75</td>
</tr>
<tr>
<td>6.6.2 Self-SL Contribution on Clustering</td>
<td>76</td>
</tr>
<tr>
<td>6.6.3 Analysis in Limited Label Regime</td>
<td>76</td>
</tr>
</tbody>
</table>
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.4 Varying Number of the Clusters</td>
<td>77</td>
</tr>
<tr>
<td>6.6.5 Results on ImageNet</td>
<td>78</td>
</tr>
<tr>
<td>6.6.6 Limitation, discussion and Future Work</td>
<td>78</td>
</tr>
<tr>
<td>6.7 Conclusion</td>
<td>79</td>
</tr>
</tbody>
</table>

References | 80   |
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>The labels of unsupervised nodes in the graph are predicted and then the graph is used to train the network.</td>
</tr>
<tr>
<td>3.2</td>
<td>Visualizations of training data which are partially labeled for (a) MNIST and (b) NORB datasets using our GSSL method.</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of Grad-CAM generated for (a) NORB and (b) MNIST datasets from our GSSL; it is shown that highlighted regions are activated by Grad-CAM algorithm for different classes.</td>
</tr>
<tr>
<td>4.1</td>
<td>Hierarchical structure for weakly supervised learning.</td>
</tr>
<tr>
<td>4.2</td>
<td>Indicates our deep framework, the CNN output is fanned out into two separate branches for fine and coarse image classification.</td>
</tr>
<tr>
<td>4.3</td>
<td>Hyper-parameters tuning: accuracy of the model on the validation set, where $S_{\text{Coarse}} =</td>
</tr>
<tr>
<td>4.4</td>
<td>Feature visualization (a) and (b); visualization of affinity matrices for coarse categories (c) and (d).</td>
</tr>
<tr>
<td>4.5</td>
<td>Affinity matrices for fine categories.</td>
</tr>
<tr>
<td>4.6</td>
<td>Example of Grad-CAM for classifying different species.</td>
</tr>
<tr>
<td>5.1</td>
<td>At each epoch, a small amount of unlabeled data is processed through the current CNN and clustered into $k$ groups. Then, the Wasserstein-2 distance is computed between these groups and the ones formed by the labeled data. Next, a regularized OT is used to form an optimal coupling between the groups from the unlabeled data and the labeled ones, using the Wasserstein-2 distance as cost function (i.e., hierarchical OT). Finally, this coupling provides pseudo-labels for the selected unlabeled data to perform a gradient descent step on the CNN. Here, circles represent unlabeled data and triangles show the labeled data and their color indicate their labels.</td>
</tr>
<tr>
<td>5.2</td>
<td>Validation error for hyperparameter tuning on CIFAR-10.</td>
</tr>
<tr>
<td>5.3</td>
<td>Indicate the number of accurate predicted labels by HrOT and CNN.</td>
</tr>
<tr>
<td>5.4</td>
<td>(a, b) OT cost trend, (c, d) mapping measures to barycenters and each other.</td>
</tr>
<tr>
<td>5.5</td>
<td>Validation error for different clustering resolution.</td>
</tr>
<tr>
<td>5.6</td>
<td>Performance of HrOT by varying the labeled data.</td>
</tr>
<tr>
<td>6.1</td>
<td>An illustration of mapping clusters to classes.</td>
</tr>
<tr>
<td>6.2</td>
<td>Number of accurate predicted labels by SSWPL in case (1) and (2).</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

6.3 Test error rate of SSWPL by changing the number of labeled data . . . . . . 75
6.4 Validation error rate of the SSWPL by varying number of clusters . . . . . . 77
List of Tables

3.1 Comparing GSCNN error rate in different scenarios on MNIST, SVHN, NORB and CIFAR-10 datasets .................................................. 22
3.2 GSCNN error rate by given number of initially labeled data per class. Results are averaged on 10 times randomly split .................................................. 22
3.3 Comparing SSL models on MNIST, SVHN, NORB and CIFAR-10 datasets .................................................. 23

4.1 Comparing performance of our model in different scenarios, we fix $|S_{\text{coarse}}| = 0.5|S|$, and we set $|S_{\text{fine}}|$ to be 0.1$|S|$, 0.2$|S|$ and 0.5$|S|$ .................................................. 40
4.2 Comparing performance of our model in different scenarios, we fix $|S_{\text{fine}}| = 0.5|S|$, and we set $|S_{\text{coarse}}|$ to be 0.1$|S|$, 0.2$|S|$ and 0.5$|S|$ .................................................. 41
4.3 The model accuracy by using and not using coarse data .................................................. 41
4.4 Comparing our method with others by fixing $|S_{\text{coarse}}| = 0.5|S|$, and setting $|S_{\text{fine}}|$ to be 0.1$|S|$, 0.2$|S|$ and 0.5$|S|$ .................................................. 42

5.1 Comparing test error between HrOT and different baselines and SSL methods .................................................. 56
5.2 Comparing test error between HrOT and different baselines and SSL methods .................................................. 58

6.1 Comparing test error between SSWPL and different baselines and SSL methods .................................................. 72
Chapter 1

Introduction

This chapter provides an introduction to the contributions presented in this report. A conceptual background is provided to give context to the underlying goals and contributions. A contextual description and literature review is provided for the fundamental technical concepts.

1.1 Problem and Motivation

Recent developments in CNNs have provided promising results for many applications in machine learning and computer vision [1–8]. However, the success of CNN models requires a vast amount of well-annotated training data, which is not always feasible to perform manually [9,10]. There are essentially two different solutions that are usually used to deal with this problem: 1) Transfer Learning (TL) and 2) Semi-Supervised Learning (SSL). In TL methods [11], the learning of a new task is improved by transferring knowledge from a related task which has already been learned. SSL methods [12], however, are motivated by the fact that in a lot of applications [13–15], there is a vast amount of unlabeled data but only a small amount of labeled data. As such, SSL methods tend to learn discriminative models that can make use of the information from an input distribution that is given by a large amount of unlabeled data. To make use of unlabeled data, one must presume that the underlying distribution of data has some structure. SSL algorithms make use of at least one of the following structural assumptions: continuity, cluster, or manifold [16]. In the
continuity assumption, data points which are close to each other are more likely to belong to the same class. In the cluster assumption, data tends to form discrete clusters, and data in the same cluster are more likely to share the same label. In the manifold assumption, data lies approximately on a manifold of much lower dimension than the input space which can be classified by using distances and densities defined on the manifold. In all of the SSL models based on these criteria, labeled and unlabeled data belonging to the same class are supposed to come from the same underlying distribution [16].

1.2 Outline and Contributions

In Chapter 2, we briefly overview state-of-the-art methods in SSL for deep learning models, as well as deep learning models for fine-image classification in weakly supervised learning fashion. We relate our approach tho these approaches and clearly explain the differences and the contributions that we have brought into this field.

In Chapter 3, we propose a novel approach to train a CNN model in SSL fashion using graph-theory. we introduce a new iterative Graph-based Semi-Supervised Learning (GSSL) method to train a CNN-based classifier using a large amount of unlabeled data and a small amount of labeled data. In this method, we first construct a similarity graph in which the nodes represent the CNN features corresponding to data points (labeled and unlabeled) while the edges tend to connect the data points with the same class label. In this graph, the missing label of unsupervised nodes is predicted by using a matrix completion method based on rank minimization criterion. In the next step, we use the constructed graph to calculate triplet regularization loss which is added to the supervised loss obtained by initially labeled data to update the CNN network parameters.

In Chapter 4, we propose a novel approach based on low rank and sparse representation of the data to leverage coarse images to improve CNN models for fine-image classification task. Specifically, we address the challenge of fine image classification in a weakly supervised fashion, whereby a subset of images is tagged by fine labels (i.e., fine images), while the remaining are tagged by coarse labels (i.e., coarse images). We propose a new deep model that leverages coarse images to improve the classification performance of fine images within
the coarse category. Our model is an end-to-end framework consisting of a Convolutional Neural Network (CNN) which uses fine and coarse images to tune its parameters. The CNN outputs are then fanned out into two separate branches such that the first branch uses a supervised low rank self-expressive layer to project the CNN outputs to the low rank subspaces to capture the global structures for the coarse classification, while the other branch uses a supervised sparse self-expressive layer to project them to the sparse subspaces to capture the local structures for the fine classification. Our deep model uses coarse images in conjunction with fine images to jointly explore the low rank and sparse subspaces by sharing the network parameters during the training which causes the data obtained by the CNN to be well-projected to both sparse and low rank subspaces for classification.

In Chapter 5, we consider the general setting of the SSL problem for image classification, where the labeled and unlabeled data come from the same underlying distribution. We propose a new SSL method that adopts a hierarchical Optimal Transport (OT) technique to find a mapping from empirical unlabeled measures to corresponding labeled measures by leveraging the minimum amount of transportation cost in the label space. Based on this mapping, pseudo-labels for the unlabeled data are inferred, which are then used along with the labeled data for training the CNN. We evaluated and compared our method with state-of-the-art SSL approaches on standard datasets to demonstrate the superiority of our SSL method.

Finally in Chapter 6, our goal is to use Wasserstein metric to provide pseudo labels for the unlabeled images to train a CNN in an SSL manner for the classification task. The basic premise in our method is that the discrepancy between two discrete empirical measures (e.g., clusters) which come from the same or similar distribution is expected to be less than the case where these measures come from completely two different distributions. In our proposed method, we first pre-train our CNN using a self-supervised learning method to make a cluster assumption on the unlabeled images. Next, inspired by the Wasserstein metric which considers the geometry of the metric space to provide a natural notion of similarity between discrete empirical measures, we leverage it to cluster the unlabeled images and then match the clusters to their similar class of labeled images to provide a pseudo label for the data within each cluster. We have evaluated and compared our method with state-of-the-art
SSL methods on the standard datasets to demonstrate its effectiveness.
Chapter 2

Semi-Supervised Learning Approaches in Deep Learning, and Backgrounds

2.1 Introduction

Training a CNN model relies on large annotated datasets, which are usually tedious and labor intensive to collect [17]. Conventional signal processing algorithms, basically apply mathematical linear and nonlinear modeling or a combination of them, to model complex 1-D signals such as EEG signal or genomic data or 2-D signals such as image data [18–22]. Two approaches are usually considered to address this problem: Transfer Learning (TL) and Semi-Supervised Learning (SSL). In TL [23], learning of a new task is improved by transferring knowledge from a related task which has already been learned. However, in SSL [24], learning of a new task is improved by using information from an input distribution that is provided by a large amount of unlabeled data. To make use of the unlabeled data, it is assumed that the underlying distribution of this data follows at least one of the following structural assumptions: continuity, clustering, or manifold [16]. In the continuity assumption [25–27], data points close to each other are more likely to belong to the same class. In the clustering assumption [25, 28, 29], data tend to form discrete clusters, and data in the same cluster are more likely to share the same label. In the manifold assumption [30, 31], data lie approximately on a manifold of much lower dimension than the input space which can be classified by distances between probability measures on the manifold [32]. There are a wide variety of SSL
algorithms in the literature [16,33]. Among all of these approaches, there are several group of methods that solely augment an additional loss term to the total loss of the SSL model over the course of training, and leave the model and training unchanged if all of the data are labeled [34–38, 38–42]. Overall, these methods are categorized into three groups including consistency regularization, entropy minimization, and pseudo labeling.

## 2.2 Consistency Regularization

Consistency regularization can be considered as a way of using unlabeled data to explore a smooth manifold on which all of the data points lie [30]. This simple criterion has provided a set of methods that are currently considered state-of-the-art for the SSL challenge. Some of these methods are Stochastic Perturbations [43], Π-Model [41], Mean Teacher [44], and Virtual Adversarial Training (VAT) [45]. The original idea behind Stochastic Perturbations and Π-Model was first introduced in [46] and has been referred to as Pseudo-Ensembles. Pseudo-Ensembles regularization techniques are usually designed such that the prediction of the model ideally should not change significantly if the data given to the model perturb; in other words, under realistic perturbations of a data point $x$ ($x \rightarrow x'$), output of the model $f_\theta(x)$ should not change significantly. This goal is achieved by adding a weighted loss term such as $d(f_\theta(x), f_\theta(x'))$ to the total loss of the model $f_\theta(x)$, where $d(\ldots)$ is mean squared error or Kullback-Leibler divergence which measures a distance between outputs of the prediction function. The main problem of Pseudo-Ensemble methods, including Π-Model is that they rely on a potentially unstable target prediction, which can immediately change during the training phase.

To address this problem, two methods Temporal Ensembling [41] and Mean Teacher [44], were proposed to obtain a more stable target output $f'_\theta(x)$. Specifically, Temporal Ensembling uses an exponentially accumulated average of outputs, $f_\theta(x)$, to make the target output smooth and consistent. Inspired by this method, Mean Teacher instead uses a prediction function which is parametrized by an exponentially accumulated average of $\theta$ during the training. Like the Π-Model, Mean Teacher adds a mean squared error loss $d(f_\theta(x), f'_\theta(x))$ as a regularization term to the total loss function for training the network. It has been shown that
Mean Teacher outperforms Temporal Ensembling in practice [44]. Contrary to Stochastic Perturbation methods which rely on constructing \( f_\theta(x) \) stochastically, VAT in the first step approximates a small perturbation \( r \) to add it to \( x \) which significantly changes the prediction of the model \( f_\theta(x) \). In the next step, a consistency regularization technique is applied to minimize \( d(f_\theta(x), f_\theta(x + r)) \) with respect to the parameters of the model, \( \theta \).

2.3 Entropy Minimization

The SSL methods based on entropy minimization uses a loss term which is applied on the unlabeled data to force the model \( f_\theta(x) \) to produce confident predictions (i.e., low-entropy) for all of the samples, regardless of what the actual labels are [47]. For example, by assuming the softmax layer of a CNN has \( c \) outputs, the loss term applied on unlabeled data is as follows:

\[
- \sum_{i=1}^{c} f_\theta^{(i)}(x) \log f_\theta^{(i)}(x).
\]

Ideally, this class of methods penalize the decision boundary that passes near the data points, while they instead force the model to provide a low-confidence prediction [47]. It has been shown that entropy minimization on its own, can not produce competitive results [48]. However, entropy minimization can be used in conjunction with VAT to provide state of the art results [45].

2.4 Pseudo Labeling

Pseudo labeling is one of the simple and straightforward approaches whereby a model incorporates it’s own predictions on unlabeled data in order to obtain additional information during the training [36,49,50]. The main downside of these methods is that they are unable to correct their own mistakes, when predictions of the model on unlabeled data are confident but incorrect. Therefore, the erroneous data can not contribute to the training and error of the models is amplified during the training. This effect is aggravated when domain of the unlabeled data is different from that of labeled data. Note that pseudo labeling is roughly similar to entropy regularization [51], in the sense that it forces the model to provide higher confidence predictions for unlabeled data [50]. However, it differs because it only forces these criteria to data points which have a low entropy prediction due to the threshold of confidence.
2.5 Deep Semi-Supervised Learning

Deep learning models with SSL algorithms have been attempted by several groups \cite{36,40,42}. Most of deep SSL approaches leverage the idea of adversarial training; however, these approaches suffer from a range of disadvantages including training instability, lack of topology generalization, and computational complexity \cite{52}. Salimans et al. \cite{39} use a model to improve the effectiveness of Generative Adversarial Networks (GAN) for SSL applications. The model provides a new technique in which the performance of supervised task is improved by learning on additional unlabeled samples. The model consists of two deep networks which are trained jointly as in typical GAN framework. The first network is a generative model that generates new samples while the other network is the discriminative network. The main problem of this method is training instability, and extra time and memory cost spent to train the two deep networks. Rasmus et al. \cite{36} merge supervised with unsupervised learning methods using a deep learning model. The model is trained to minimize the sum of supervised and unsupervised cost functions by using back propagation, and at the same time prevents the need for layer-wise pre-training. The main problem of this method is the lack of a clear path to generalize it to other network topologies, such as recurrent or residual networks. The probabilistic formulation of CNN models proposed in \cite{53} natively supports SSL introduced by using a new family of hierarchical generative models. However, the main concern of these methods is that the activation function requires to be ReLU and that the overall network topology follow a CNN. There are some other methods which extend the generative models for the SSL; for example, Maaløe et al. \cite{38} extend GANs with auxiliary variables to learn better variational approximations and more expressive variational distribution. Tobias et al. \cite{37} propose categorical GAN (CatGAN) which replaces the binary discriminator in the standard GAN with a multi-class classifier, and trains the generator and the discriminator using mutual information on unlabeled samples. Kingma et al. \cite{34} use conditional Variational Autoencoders (VAEs) to treat labels as conditions of generative models to describe the input data; they make posterior inference of labels given unlabeled samples to generate a particular class of samples.
2.6 Graph-Based Semi-Supervised Learning

The GSSL approaches generally contain two main steps. In the first step, the graph is constructed from all the data points (both labeled and unlabeled data) to represent the relationship between them while in the second step, the information from the labeled data is propagated to the unlabeled data over the graph. Among the different GSSL methods which formulate the information propagation step by using different objective functions such as low-rank minimization [54], k-nearest neighbor methods [55], structured sparsity [56], min-cut [57], energy minimization [58] and Laplacian spectral method [13], there is one common assumption which states that the data points on the same structure (i.e., manifold, cluster or subspace) more likely have the same label. In fact, GSSL methods tend to model the structural density among the data points by measuring the proximity (similarity) between data points in the graph and then propagate information of labeled data to the unlabeled data in a way that the missing labels in the graph are predicted based on the closest labeled data points (e.g., k-nearest neighbors classifier). Since normally there is no explicit solution to model the underlying structures of the data in the feature space, a graph created from the data usually serves as an approximation of the real structure. As a result, constructing a proper graph that best captures the main structure of the data point is important to all GSSL approaches [59].

2.7 Hierarchical Structure for Visual Recognition

The hierarchical structure between objects in most large-scale datasets, such as ImageNet, has been incorporated in deep models to learn each category of images in conjunction with the other categories to improve the overall recognition performance [60–65]. Most of these methods such as [64], learn the shared and disjoint properties among the objects jointly by focusing on their commonalities and differences in a class hierarchy such that the shared properties discriminate the objects at coarse level of abstraction, while the disjoint properties characterize them at fine level of abstraction. For example, Srivastava et al. [61] create a class hierarchy and use a CNN model which transfers the knowledge between the classes to
enhance the overall performance by using a small number of training samples.

In other scenarios, Xiao et al. [63] introduce a training method that expands a network hierarchically. Due to scalability constraints, the categories in this method are first grouped together based on their similarities, and then self-organized into two groups including coarse and fine levels.

Furthermore, Goo et al. [60] propose a method that uses the shared and specialized properties in a semantic hierarchical structure to learn improved discriminative CNN features. This method uses min and difference pooling to implement generalization and specialization layers. In another case, Guo et al. [66] introduce an end-to-end framework that integrates a CNN and a Recurrent Neural Network (RNN) for hierarchical categorization. In this method, goal of the CNN is to obtain discriminative features from the input images, while the goal of the RNN is to train the coarse and fine image classification jointly.

Among all of the aforementioned methods which use hierarchical structure to improve fine and coarse classification performance, there are only two methods [64,66] that consider fine image classification in a weakly supervised fashion. Here, we revisit the hierarchical structure between the fine and coarse categories and propose a new deep model which uses the self-expressiveness property of the data with low rank and sparse representation to tackle the challenge of fine image classification in a weakly supervised fashion.

### 2.8 Preliminaries, and Background

#### 2.8.1 Matrix Completion Based on Rank Minimization

The problem of completing a low-rank matrix from a few sampled entries has been successfully applied in a variety of applications such as the Netflix challenge. A major breakthrough by Candes et al. [67] states that minimizing a matrix rank subject to some constrains can be recast as minimizing the nuclear norm (sum of singular values) of the matrix. Since nuclear norm minimization of a matrix has characteristic of a Semidefinite Programming (SDP), many approaches have been proposed to solve this minimization problem effectively [68]. In the field of computer vision and machine learning, nuclear norm
minimization has been applied to many problems such as robust PCA \cite{69} and subspace segmentation \cite{70}.

2.8.2 Low Rank and Sparse Representation

Low Rank and Sparse Representation (LRR, SR) methods are usually used in an unsupervised manner to capture low dimensional linear subspaces underlying the data \cite{70, 73}. These subspaces usually have a self-expressive property, meaning that a sample taken from a single subspace can be expressed as a linear combination of other samples from the same subspace \cite{73, 74}. Generally, in LRR and SR, an affinity matrix is constructed to measure the pairwise similarities between the data points. The LRR methods usually construct the affinity matrix such that it has the minimum possible rank, while SR methods construct the affinity matrix such that it has the minimum $\ell_1$ norm.

There are also supervised versions of LRR \cite{75, 76} and SR \cite{77, 81}. These methods sufficiently exploit the labeled data to learn a discriminative low rank and sparse representation for the data points. These methods incorporate label information as a constraint to guide the learning process for exploring a robust and discriminative subspace projection \cite{82, 83}. In these methods, the data from different classes are well-separated after projection. Like the supervised methods, in this work we incorporate the label information by using a contrastive loss function \cite{84} during the projection of the CNN outputs to the low rank and sparse subspaces to increase the class separability.

2.8.3 Low Rank and Sparse Self-Expressive

Assume that the data points $\{x_1, x_2, ..., x_n\}$ are clean and sampled from multiple linear subspaces. A subspace is considered self-expressive if each data point from the subspace is expressed by a linear combination of the other data from the same subspace. By stacking all of the data points $x_i$ in a column-wise fashion into a data matrix $X$, this property can be represented by a linear equation as follows: $X = XC$, where $C$ is an affinity matrix which measures the pairwise similarity between all of the data points. Specifically, this idea is
formulated by an optimization problem as:

$$\min_C ||C||_p \quad \text{s.t.} \quad X = XC, \quad \text{diag}(C) = 0,$$

(2.1)

where $p$ is an arbitrary matrix norm. The diagonal constraint on $C$ avoids trivial solutions such as identity matrix.

Various methods have been introduced to define an affinity matrix to explore a set of subspaces. All of these methods aim to find $C$ such that $c_{ij} \neq 0$ if $x_i$ and $x_j$ are in the same subspace, and $c_{ij} = 0$ if they are in different subspaces. In the low rank representation $[70, 71, 85]$, the nuclear norm of $C$ is minimized in (1) instead of minimizing the rank of $C$, because rank minimization is an NP hard problem and the nuclear norm is the tightest convex relaxation to the rank $[86]$. Following the model $[71]$ for LRR in general case which data may be contaminated by noise, the model searches for a clean dictionary (e.g., $A$), and then assumes that the data are obtained by adding noise or error term (e.g., $E$) to the clean dictionary (i.e., $X = A + E$). In this theoretically sound model, the error term for LRR is relaxed by a Frobenius norm as follows:

$$\min_C ||C||_* + \frac{\lambda}{2} ||X - XC||_F^2,$$

(2.2)

where, $||.||_*$ is the nuclear norm and $||.||_F$ is the Frobenius norm, and $\lambda$ is a hyper-parameter that balances the nuclear and the Frobenius norm. By denoting the SVD of $X$ as $U\Sigma V$, the optimal solution of (2) is obtained by using:

$$\hat{C} = V_1(I - \frac{1}{\lambda} \Sigma^{-2})V_1^T,$$

(2.3)

where $U = [U_1, U_2]$, $\Sigma = \text{diag}(\Sigma_1, \Sigma_2)$ and $V = [V_1, V_2]$. Matrices are partitioned according to the sets $I_1 = \{i : \sigma_i > \frac{1}{\sqrt{\lambda}}\}$ and $I_2 = \{i : \sigma_i \leq \frac{1}{\sqrt{\lambda}}\}$ $[71]$.

In a sparse representation, however, each data point is expressed by the minimum possible number of the other data from the same subspace. Similar to the low rank representation case, we relax the optimization problem by replacing the $\ell_0$ with the $\ell_1$ norm, because the $\ell_0$ norm is a combinatorial norm and $\ell_1$ is the closest convex norm to the $\ell_0$ norm. Thus, (1) for
the sparse subspace, is as follows:

$$\min_C \|C\|_1 + \frac{\lambda}{2}\|X - XC\|_F^2, \quad \text{diag}(C) = 0.$$  \hspace{1cm} (2.4)

This problem can be efficiently solved by Alternating Direction Method of Multipliers (ADMM) algorithm \cite{87}.
Chapter 3

Matrix Completion for Graph-Based Deep Semi-Supervised Learning

Convolutional Neural Networks (CNNs) have provided promising achievements for image classification problems. However, training a CNN model relies on a large number of labeled data. Considering the vast amount of unlabeled data available on the web, it is important to make use of these data in conjunction with a small set of labeled data to train a deep learning model. In this chapter, we introduce a new iterative Graph-based Semi-Supervised Learning (GSSL) method to train a CNN-based classifier using a large amount of unlabeled data and a small amount of labeled data. In this method, we first construct a similarity graph in which the nodes represent the CNN features corresponding to data points (labeled and unlabeled) while the edges tend to connect the data points with the same class label. In this graph, the missing label of unsupervised nodes is predicted by using a matrix completion method based on rank minimization criterion. In the next step, we use the constructed graph to calculate triplet regularization loss which is added to the supervised loss obtained by initially labeled data to update the CNN network parameters.

3.1 Introduction

CNN models require vast amounts of labeled data to be trained properly; however, providing reliable annotated data to train the CNN models tends to be expensive [88]. There
are essentially two principal solutions that are usually used to deal with this challenge: 1) Transfer Learning (TL) and 2) Semi-Supervised Learning (SSL). In TL methods [89], we enhance new task learning via transfer of knowledge from a related task which has already been learned. In SSL methods [90], however, we are motivated by the fact that in a lot of applications, there are a vast amount of unlabeled data but only a small amount of labeled data and we essentially aim to learn discriminative learning methods that can make use of the information about the input distribution that is given by a large amount of unlabeled data. The SSL is a broad research field which has been used in variety of applications such as image search [13], biometric template security [91,92], and natural language processing [93]. Among the recent SSL approaches, the GSSL methods have received a lot of attention and have become popular due to their flexibility in practical applications and low computational complexity. In GSSL methods, one assumes that the data points (both labeled and unlabeled) are embedded in a low-dimensional manifold which might be reasonably represented by a graph. In GSSL methods, each data point is expressed as a node in a graph and weights between nodes provide a measure of similarity between them. In GSSL, we inject seed labels on a subset of the nodes and then we infer labels on the unlabeled nodes in the graph. The intuition behind the similarity graph is that it captures the information from the labeled samples which is then propagated through to the unlabeled samples within the graph.

In this chapter, we propose a novel iterative GSSL algorithm to train a CNN-based classifier. Our GSSL algorithm uses a new method to construct a similarity graph by leveraging matrix completion method based on rank minimization criteria. Once the similarity graph is constructed, we use it to regularize the fully supervised loss (i.e., given by initially labeled data points) to force that connected data points in the graph (i.e., data points which belong to the same class) share similar feature representations while disconnected ones have different representations.

The entire framework is trained end to end such that in each training iteration, the feature representations computed from the CNN are used to construct the similarity graph, then the graph is used to calculate triplet regularization loss which is added to the supervised loss to update the parameters of the network which provides new feature representations for the next iteration.
3.2 Similarity Graph Construction

In this section, we describe our method to construct the similarity graph via the matrix completion method. Let \( g(x_i) \in \mathbb{R}^d \) be the features captured by the CNN model corresponding to \( n \) samples; each of these samples is represented by a node in the similarity graph. Let \( X = [g(x_1), g(x_2), ..., g(x_n)] \) be a \( d \times n \) feature matrix constructed by stacking samples column wise. Suppose that \( c \) is the number of classes; \( y_1, ..., y_n \) are one hot encoding label of samples, and \( Y = [y_1, ..., y_n] \) indicates a \( c \times n \) label matrix which is obtained by a linear model from \( X \) (i.e., \( y_i = Wg(x_i) + b \), where \( W \) is a \( c \times n \) weight matrix). In our problem, all the entries in feature matrix \( X \) are known and observable; however, entries corresponding to unlabeled samples in the label matrix \( Y \) are missed and we essentially aim to predict them. We note that by assuming \( X \) as a low rank matrix, the combined \( (c + d) \times n \) matrix \( Z = [X; Y] \) produces a low rank matrix too (i.e., \( \text{rank} ([X; Y]) \leq \text{rank} (X) + c \)); and predicting the missing entries in this matrix can be cast as a matrix completion problem \cite{94,95}. Here, we take advantage of the matrix completion method to predict missing labels in the graph.

Similar to other GSSL methods, we assume that the data points are embedded within a low-dimensional manifold; indeed we make a structural assumption about the data points. This assumption causes the feature matrix \( X \) to have a low rank or approximately have a low rank, and consequently it provides the fact that the matrix \( Z \) should also have a low rank. Therefore, we set up an optimization problem to predict missing labels in the graph where the decision variable \( Z \) is the data matrix with the missing labels. We aim to find the matrix \( Z \) which matches the observed entries (i.e., labels of supervised data and matrix \( X \)) and has the minimum rank.

However, rank minimization is an NP hard problem. To the best of our knowledge, the best algorithm we know to minimize the rank of a matrix under linear constraint is doubly exponential in the size of the matrix \cite{96}, which means that when the size of the matrix is more than seven, it is not practical to solve it on the computer. To relax the rank term in our optimization problem, we use the heuristic proposed in Candes et al. \cite{67}; the heuristic uses an alternative method which minimizes the nuclear norm \( \|Z\|_* = \sum_{k=1}^n \sigma_k(Z) \) (the sum of the singular values) under the constraint set over the observed entries; where \( \sigma_k(Z) \) denotes
the $k^{th}$ largest singular value of $Z$. The nuclear norm of matrix $Z$ is the dual norm of the spectral norm of the matrix $Z$ which is convex, and it can be solved by variety of convex optimization algorithms $^{68}$. The relationship between rank and nuclear norm is similar to that of $\ell_0$ norm and $\ell_1$ norm for vectors. Since we can not minimize the rank of the matrix, we choose the nuclear norm as an alternative optimization problem which is the tightest convex relaxation to the rank.

### 3.2.1 Predicting Missing Labels via Matrix Completion

In this section, we provide an optimization problem to predict missing entries of the data matrix $Z = [X; Y]$ such that the nuclear norm of $Z$ is minimized and entries in $Z$ match the observed entries (i.e., labels of supervised data and feature matrix $X$). Let $\Omega_X$ be the index set of observed entries in the feature matrix $X$, where $(i, j) \in \Omega_X$ if and only if $X_{ij}$ is observed. Likewise, let $\Omega_Y$ be the index set of observed entries in the label matrix $Y$ and $(i, j) \in \Omega_Y$ if and only if $y_j$ is a labeled sample. In this problem $|\Omega_X| = d \times n$ (no missing entries in $X$), and $1 < |\Omega_Y| < c \times n$ (some missing entries in $Y$). The optimization problem for predicting missing entries in $Z$ is defined as follow:

$$
\text{argmin}_Z \mu \|Z\|_* + \frac{1}{|\Omega_X|} \sum_{i,j \in \Omega_X} c_x(z_{ij}, x_{ij}) + \frac{\lambda}{|\Omega_Y|} \sum_{i,j \in \Omega_Y} c_y(z_{(i+d)j}, y_{ij}).
$$

(3.1)

We shift row index of the stacked matrix $Z$ in the $c_y(z_{(i+d)j}, y_{ij})$ because we want to skip $X$ part in the $Z$. Apart from minimizing the nuclear norm of $Z$, we penalize the cost function in (1) by adding $c_y(.)$ and $c_x(.)$ losses to avoid trivial solutions and large distortions of $Z$ from the observed entries in $X$ and $Y$ matrices. The observed label data type is of a different type than the observed feature data; thus we define two different losses. The $c_x(z_{ij}, x_{ij}) = \frac{1}{2}(z_{ij} - x_{ij})^2$ is defined as the squared loss, while the $c_y(z_{(i+d)j}, y_{ij}) = \log(1 + \exp(-z_{(i+d)j}y_{ij}))$ is the logistic loss which accentuates the error on entries switching labels as is different from their absolute numerical deviation. The parameters $\mu, \lambda$ are the weights which create a balance between errors for better label error correction and feature adaptation.
Optimization Method

The loss defined in (1) is a convex optimization problem. We use the soft-impute algorithm \[97\] which is a simple and effective algorithm for nuclear norm regularized matrix completion. This algorithm iteratively restore the missing entries with those attained from a soft-thresholded SVD. We first define a projection operator \(P_{\Omega}\) on the observed set \(\Omega\) as follows:

\[
[P_{\Omega}(Z)]_{ij} = \begin{cases} 
z_{ij} & (i, j) \in \Omega \\
0 & (i, j) \notin \Omega
\end{cases}
\]  

(3.2)

thus, the optimization problem in (3) can be rewritten as follows:

\[
f(Z) = \underbrace{\mu ||Z||_*}_{G(Z)} + \underbrace{\frac{1}{2||X||}||P_{\Omega}(X) - P_{\Omega}(Z_x)||_F^2}_{H(Z_x)} + \underbrace{\lambda \frac{|\Omega_Y|}{|\Omega|} \log(1 + \exp(-P_{\Omega}(Z_y) \circ P_{\Omega}(Y)))}_{U(Z_y)},
\]

(3.3)

where, \(Z_x\) and \(Z_y\) are sub-matrices of \(Z\) for parts \(X\) and \(Y\), respectively (i.e., \(Z = [X; Y]\)), and symbol \(\circ\) indicates Hadamard or element wise product between \(P_{\Omega}(Z_y)\) and \(P_{\Omega}(Y)\). In this set of formulation, we call the second and third parts \(H(Z_x), U(Z_y)\), respectively which are convex and smooth, and the first part \(G(Z)\) which is also convex but not smooth. Therefore, we can think about three ingredients needed for proximal gradient descent:

- The first is \(\nabla H(Z_x)\); here subgradient is just the gradient:

\[
\nabla H(Z_x) = -\frac{1}{|\Omega_X|}(P_{\Omega}(X) - P_{\Omega}(Z_x)),
\]

(3.4)

where, the gradient of \(H(z_{ij})\) is \(-\frac{1}{|\Omega_X|}(x_{ij} - z_{ij})\).

- The second is \(\nabla U(Z_y)\); here, subgradient is also just the gradient:

\[
\nabla U(Z_y) = \frac{\lambda}{|\Omega_Y|} \frac{-P_{\Omega}(Y)}{1 + \exp(P_{\Omega}(Z_y) \circ P_{\Omega}(Y))},
\]

(3.5)

where, the gradient of \(U(z_{(i+d)j})\) is \(\frac{-y_{ij}}{1 + \exp(z_{(i+d)j}y_{ij})}\).
• The third is the \textit{prox} operator:

\[
\text{prox}_t(Z) = \arg \min_C \frac{1}{2t} ||Z - C||_F^2 + \mu||C||_*. \tag{3.6}
\]

It can be proved that \(\text{prox}_t(Z) = S_\lambda(Z)\) \cite{97} which is the \textit{matrix soft thresholding at the level} \(\lambda\); where \(S_\lambda(Z)\) is defined by

\[
S_\lambda(Z) = U\Sigma_\lambda V^T, \tag{3.7}
\]

where, \(Z = U\Sigma_\lambda V^T\) is an SVD, and \(\Sigma_\lambda\) is diagonal with \((\Sigma_\lambda)_{ii} = \max\{\Sigma_{ii} - \lambda, 0\}\). This matrix soft-thresholding is an element-wise soft-thresholding of the matrix. By using a soft-threaded singular matrix \(\Sigma_\lambda\), this returns a low-rank \textit{prox} result.

Therefor, the Proximal Gradient update step is written as follows:

\[
Z^+ = \text{prox}_t(Z - t(\nabla H(Z_x) + \nabla U(Z_y)))) = S_\lambda(Z - t(\nabla H(Z_x) + \nabla U(Z_y))). \tag{3.8}
\]

Note that \((\nabla H(Z_x) + \nabla U(Z_y))\) is Lipschitz continuous with \(L = 1\), thus we can choose fixed step size \(t = 1\) and then the update step in this case is expressed as follows:

\[
Z^+ = S_\lambda(Z - (\nabla H(Z_x) + \nabla U(Z_y))). \tag{3.9}
\]

Once the label of unlabeled data are predicted by matrix completion; in the next step, the nodes in the graph are connected to each other if they belong to the same class and they are disconnected otherwise.

### 3.3 Supervised Task Regularization via the Constructed Graph

In this section, we provide a new approach based on the triplet loss function to leverage the constructed graph for training the CNN model. In this approach, we aim to regularize the supervised task by adding a semi-supervised loss term as an auxiliary task to the CNN. In other words, we concentrate mostly on the classifier regularization learned in a supervised
fashion with few labeled data. To regularize the supervised task using unsupervised data, we apply the triplet loss function using a triplet of data on the graph as follows;

\[
\mathcal{L}_{\text{trip}}(g(x_a), g(x_p), g(x_n), A) = \\
\max(||g(x_a) - g(x_p)||^2 - ||g(x_a) - g(x_n)||^2 + \alpha, 0),
\]

(3.10)

where \(A\) is the adjacency matrix of the graph in which entries in the matrix indicate if pairs of the nodes are adjacent or not in the graph; the \(\alpha\) parameter is the margin in the triplet loss and \(g(x_a), g(x_p), g(x_n)\) are the output of CNN for the images \(x_a, x_p\) and \(x_n\), respectively.

In triplet loss, we look at the three data point on the graph at the same time; in our case we choose labeled samples in the graph as the anchor (i.e., sample \(x_a\) in (10)) and sample \(x_p\) as a positive sample if \(A(x_a, x_p) = 1\) and sample \(x_n\) as a negative sample if \(A(x_a, x_n) = 0\); we want to bring positive and anchor pairs (i.e., two images which are connected in the graph) close to each other while push away the negative and anchor pairs (i.e., two images which are not connected in the graph) simultaneously. The triplet loss is added to the total network loss function to regularize the supervised classification loss with an auxiliary semi-supervised loss term.

### 3.3.1 Deep Semi Supervised Loss Function

Now, we have all the loss terms including the supervised loss and semi-supervised loss to set up the total CNN loss function. Our SSL loss function for updating the CNN parameters is defined as follows:

\[
\mathcal{L}(w, x, y) = \sum_{x_i \in x_s} \mathcal{L}_c(w, g(x_i), y_i) + \\
\gamma \left( \sum_{x_a \in x_s: \{x_p, x_n\} \in x} \mathcal{L}_{\text{trip}}(w, g(x_a), g(x_p), g(x_n), A) \right),
\]

(3.11)

where \(x\) and \(y\) are the training batch and samples labels in the training batch (i.e., real labels and predicted labels); \(\mathcal{L}_c\) is supervised classification loss (softmax loss is used, but can...
be chosen other type of losses such as center loss, contrastive center loss); $x_s$ is the supervised samples in training batch $x$. The $w$ is the parameters of the CNN and $\gamma$ is the balancing term between two supervised and semi-supervised losses. The batch set is created such that number of similar (i.e., anchor and positive) and dissimilar (i.e., anchor and negative) pairs to be roughly balanced. In each triplet, the labeled data (not predicted labels) are chosen as the anchor and predicted labeled data are considered only as positive and negative samples in the batch.

### 3.4 Experiments

**Datasets and Pre-Processing.** We conducted our experiments on the widely used MNIST [98], SVHN [99], small NORB [100] and CIFAR 10 [101] datasets. For each of these datasets, we split the training set to two different sets of labeled and unlabeled samples. We ensure that all the classes are balanced such that each class should have the same number of labeled samples. We ran our model for 10 times with different random splits of the labeled and unlabeled data for each dataset, and we report the mean and standard deviation of the error rate.

**MNIST** is handwritten digits of 10 different classes dataset which contains a training set of 60,000 samples, and a test set of 10,000 samples. The digits have been size-normalized and centered in a fixed-size ($28 \times 28$) images. We select 100 samples in the training set as labeled and remaining of it as unlabeled.
Figure 3.2: Visualizations of training data which are partially labeled for (a) MNIST and (b) NORB datasets using our GSSL method.

<table>
<thead>
<tr>
<th>Methods</th>
<th>MNIST (100)</th>
<th>SVHN (1000)</th>
<th>NORB (1000)</th>
<th>CIFAR-10 (4000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Completion</td>
<td>1.98(±0.03)</td>
<td>10.06(±0.08)</td>
<td>9.11(±0.11)</td>
<td>19.91(±0.23)</td>
</tr>
<tr>
<td>GSCNN+No Reg</td>
<td>1.14(±0.09)</td>
<td>8.41(±0.22)</td>
<td>8.71(±0.18)</td>
<td>18.03(±0.31)</td>
</tr>
<tr>
<td>GSCNN</td>
<td>0.84(±0.12)</td>
<td>5.13(±0.39)</td>
<td>7.01(±0.53)</td>
<td>15.49(±0.64)</td>
</tr>
</tbody>
</table>

Table 3.1: Comparing GSCNN error rate in different scenarios on MNIST, SVHN, NORB and CIFAR-10 datasets.

**SVHN** is another digit dataset similar to MNIST with $32 \times 32$ color images centered around a single character. The task is to classify the center digits in the images; we follow [102, 103] methods to split the dataset to 598,388 training data and 26,032 testing data. For this dataset, we choose randomly 1000 samples as labeled and rest of it, is used as unlabeled.

**CIFAR-10** is a collection of $32 \times 32$ RGB images of 10 classes including airplane, automobile, bird, cat, deer, dog, frog, horse, ship and trucks. This dataset contains 50,000 number of images for training and 10,000 for testing; we select 4,000 samples in the training set as labeled and rest as unlabeled.

<table>
<thead>
<tr>
<th>Samples Per Class</th>
<th>100</th>
<th>200</th>
<th>400</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORB</td>
<td>9.88(±0.54)</td>
<td>7.01(±0.53)</td>
<td>6.12(±0.41)</td>
<td>5.07(±0.19)</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>18.98(±0.62)</td>
<td>16.82(±0.47)</td>
<td>15.49(±0.64)</td>
<td>14.51(±0.34)</td>
</tr>
</tbody>
</table>

Table 3.2: GSCNN error rate by given number of initially labeled data per class. Results are averaged on 10 times randomly split.
NORB contains gray scale images of 5 general classes including animal, human, airplane, truck and car. The objects were imaged by two cameras under different lighting conditions, elevations and azimuths. This dataset contains 24,300 images for both training and testing sets. In our experiments, we resize the images to $32 \times 32$ as it is in [104]; we select 1,000 samples in the training set as labeled and remaining is considered as unlabeled.

**Experimental Setup.** Our CNN architecture has been shown in Fig.4.2; it is composed of three convolutional layers, three max pooling layers and one fully connected layer. There are 64 number of filters in the first convolutional layer and 128 number of filters in the second and third convolutional layers, respectively. The size filters in this architecture are $3 \times 3$ and the convolution stride is set to 1 pixel. To preserve spatial resolution after convolution, spatial padding of the convolutional layer is fixed to 1 pixel for all $3 \times 3$ convolutional layers. The max-pooling layers are placed after each convolutional layers, respectively; the max-pooling is carried out on a $2 \times 2$ pixel window with a stride of 2. We apply batch normalization (i.e., shifting inputs to zero-mean and unit variance) after each convolutional and fully connected layer, and before performing the Rectified Linear Units (ReLU) activation function. The batch normalization potentially helps to achieve faster learning as well as higher overall accuracy. Furthermore, batch normalization allows us to use a higher learning rate, which potentially provides another boost in speed. The parameters of the network are initialized by sampling randomly from $\mathcal{N}(0,0.001)$ except for the bias parameters which are initialized

<table>
<thead>
<tr>
<th>Methods</th>
<th>MNIST (100)</th>
<th>SVHN (1000)</th>
<th>NORB (1000)</th>
<th>CIFAR-10 (4000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1 + M_2$</td>
<td>3.33(±0.14)</td>
<td>36.02(±0.1)</td>
<td>18.79(±0.05)</td>
<td>-</td>
</tr>
<tr>
<td>VAT</td>
<td>2.33</td>
<td>24.63</td>
<td>9.88</td>
<td>-</td>
</tr>
<tr>
<td>Ladder</td>
<td>1.06(±0.37)</td>
<td>-</td>
<td>-</td>
<td>20.40(±0.47)</td>
</tr>
<tr>
<td>CatGAN</td>
<td>1.39(±0.28)</td>
<td>-</td>
<td>-</td>
<td>19.58(±0.46)</td>
</tr>
<tr>
<td>ADGM</td>
<td>0.96(±0.02)</td>
<td>22.86</td>
<td>10.06(±0.05)</td>
<td>-</td>
</tr>
<tr>
<td>SDGM</td>
<td>1.32(±0.07)</td>
<td>16.61(±0.24)</td>
<td>9.4(±0.04)</td>
<td>-</td>
</tr>
<tr>
<td>FM</td>
<td>0.93(±0.07)</td>
<td>8.11(±1.3)</td>
<td>-</td>
<td>18.63(±2.32)</td>
</tr>
<tr>
<td>Triple GAN</td>
<td>0.91(±0.58)</td>
<td>5.77(±0.17)</td>
<td>-</td>
<td>16.99(±0.36)</td>
</tr>
<tr>
<td>$\Pi$ model</td>
<td>-</td>
<td>5.43(±0.25)</td>
<td>-</td>
<td>16.55(±0.29)</td>
</tr>
<tr>
<td>ALI</td>
<td>-</td>
<td>7.42(±0.65)</td>
<td>-</td>
<td>17.99(±1.62)</td>
</tr>
<tr>
<td>GSCNN</td>
<td>0.84(±0.12)</td>
<td>5.13(±0.39)</td>
<td>7.01(±0.53)</td>
<td>15.49(±0.64)</td>
</tr>
</tbody>
</table>

Table 3.3: Comparing SSL models on MNIST, SVHN, NORB and CIFAR-10 datasets
as zero. We implemented our framework in TensorFlow and performed our experiments on two GeForce GTX TITAN X 12GB GPUs. We use Adam optimizer \cite{105} with the default hyper-parameters values (\( \epsilon = 10^{-3} \), \( \beta_1 = 0.9 \), \( \beta_2 = 0.999 \)) in our experiments. The batch size in all experiments is fixed to 128, and we set \( \gamma \) to 0.1 experimentally to create a balance between supervised loss and unsupervised loss in the total network loss function.

**Hyper-Parameter Tuning.** We used 10-fold cross validation in each experiment to tune hyper-parameters in our model. For \( \lambda \) in (1), we randomly divide the labeled data into ten disjoint subsets; next we run the matrix completion over \( \frac{9}{10} \) and we calculate the performance on the remaining \( \frac{1}{10} \); then we average the results over the 10 folds (We note that we used label error as performance criterion to select parameters because our goal in MC is to predict label of unlabeled data points). The range of \( \lambda \) values are \{\( 10^{-3}, 10^{-2}, 10^{-1}, 1 \}\}. For \( \mu \) in (1), we initialize it to be \( 0.25\sigma_1 \), where \( \sigma_1 \) is the largest singular value of the matrix \([X; Y]\) and decrease it gradually until \( 10^{-5} \) as it is suggested in \cite{97}.

**Matrix Completion.** We use SoftImput algorithm implemented in fancyimput package in python to predict missing labels in the graph. We set shrinkage value which is the value by which we shrink singular values on each iteration to the maximum singular value of the initialized matrix (zeros for missing values) divided by 100; the maximum number of SVD iteration is set to 1000. In matrix completion, we stop the algorithm by defining a convergence threshold (0.001 in our experiments) which is the minimum ration difference between iterations (as a fraction of the Frobenius norm of the current solution). In SoftImput algorithm, a sequence of solutions are produced for which the criterion decreases to the optimal solution with every iteration. Convergence threshold can be given to the SoftImput algorithm implemented in fancyimput package.

**Triplet Mining on the Graph.** In our experiments, we concentrate on the online triplet mining strategy to generate useful triplets for data on the graph. Considering a batch of \( b \) samples, we extract CNN features for each sample, and we then can create a maximum of \( b^3 \) triplets even though most of these triplets are not valid (i.e., triplet except two positive and one negative). This technique provides us more triplets for a single batch of samples during the network training. We use all the valid triplets in the training and we average the loss on the hard triplet (i.e., \( d(g(x_a), g(x_a)) < d(g(x_p), g(x_a)) \)) and semi-hard triplet (i.e.,
\[ d(g(x_a), g(x_p)) < d(g(x_a), g(x_p)) + \alpha \); we disregard the easy triplets (triplets with zero loss) because averaging on them makes the overall loss very small.

**Effectiveness of Regularization based on Triplet.** To show the effectiveness of our semi-supervised loss based on the triplet in training of our CNN model, we conducted experiments in two different scenarios, a) GSCNN+No Reg which indicates the case where we remove the triplet regularization loss in (11) and use all the labeled and unlabeled data with true and completed labels directly in the softmax classifier loss, b) GSCNN which indicates the case where we use the triplet regularization loss in (11) in training of our CNN model. Table 3.1 shows the performance of our model in two different scenarios (i.e., a) GSCNN+No Reg and b) GSCNN); the results show that our regularization loss based on the triplet can improve the model performance by 0.3%, 3.28%, 1.7% and 2.54% on the MNIST, SVHN, NORB and CIFAR-10 datasets, respectively. This improvement is because the softmax classifier loss only forces the CNN features of different classes to stay apart, while the triplet loss not only does this, but also efficiently brings the CNN features of the same class close to each other. Therefore, by considering triplet regularization in the training, not only the inter-class features differences are enlarged, but also the intra-class features variations are reduced. Moreover, since our method is an iterative process of two steps (i.e., the first step uses matrix completion to predict the labels, and the next step uses the predicted results to train a CNN), we reported in Table 3.1 the matrix completion error rates to provide an empirical analysis showing that matrix completion has significant influence on training the CNN model. Accuracy of the matrix completion indicates those unlabeled data predicted correctly by matrix completion, are then used in the regularizer to improve the CNN performance.

**Constructed Graph Properties.** The graph in our model is constructed dynamically while the CNN network is trained; this is because the graph in our model needs the network output to be constructed. In our SSL method, the graph is created online in a local scope (over a few samples of training set) which is virtually similar to the concept of training batch in the CNN models. In each training batch, the labels of the unsupervised data are predicted based on all the labeled data using our matrix completion method. Indeed, we take a batch of training data and then we predict the class label of unsupervised data to construct the
Figure 3.3: Example of Grad-CAM generated for (a) NORB and (b) MNIST datasets from our GSSL; it is shown that highlighted regions are activated by Grad-CAM algorithm for different classes.

graph for the batch. Our graph construction method is an Expectation Maximization (EM) like algorithm in a sense that in forward pass, the graph is constructed for a batch (including labeled and unlabeled data), and later on it is used as a regularizer in the network loss calculation to update the network parameters by back propagation in the backward pass. This property enables us to create a robust graph through the training step; because the graph is constructed by better set of CNN features as we train the CNN through several training epochs. This factor makes the graph construction method more robust in comparison to the offline based graph construction methods with static data embedding.

Most of graph construction algorithms are usually expensive in terms of time complexity. For example, graph construction using offline $k$-NN method in brute-force fashion is $O(n^2)$ where $n$ is number of training data. Even though there are other efficient methods \[106\] to improve $k$-NN method in terms of time complexity, in most of the offline construction methods the time complexity usually makes graph construction step unpractical specially for the large scale datasets. However, splitting the data to small chunks of data with equal size which in our case is the batch makes the graph construction step more efficient, and also feasible for online computation and simultaneous with CNN training; because in this case, only a small portion of the training data is used to construct the graph. Our method is online and use small part of the whole data. The computationally demanding part of our
graph construction algorithm is the equation (9) where we take SVD from a low-rank matrix \( Z \) to predict missing label of unsupervised nodes in the graph. For example, it takes around 6 (secs) to complete a 100 \( \times \) 100 matrix in each iteration of our algorithm.

**Effect of Number of Initially Labeled Data in the Model.** Since supervised data points are taken as anchors while forming the triplets in our model, we conducted experiment showing the influence of the number of initially labeled data points on the final performance. In this experiment, we selected 100 samples per each class and gradually increased the number of samples to 200, 400, and 800, respectively; the results in Table 3.2 indicate that the model performance increases as the number of initially labeled data points in the regularizer increases. The trend of improvement is reported in Table 3.2. The results on the CIFAR-10 dataset show the model is improved by around 2.16%, 1.33% and 0.98% when we increase the number of labeled samples from 100 \( \rightarrow \) 200, 200 \( \rightarrow \) 400 and 400 \( \rightarrow \) 800, respectively, while on the NORB dataset, the model performance is improved by around 2.87%, 0.89% and 1.05% when we increase the number of labeled samples from 100 \( \rightarrow \) 200, 200 \( \rightarrow \) 400 and 400 \( \rightarrow \) 800, respectively.

**Evaluation and Discussion.** We compare our GSSL model with a large body of previous models on MNIST, SVHN, NORB and CIFAR-10 datasets using 100, to , 4000 labeled samples, respectively. Experimental results in Table 3.3 show that our method is competitive to the state-of-the-art results for all these datasets; given 100 labeled samples on the MNIST dataset, our method still is comparable to the outstanding generative models including FM [39] and Triplet GAN [40]; Table 3.3 also shows Semi-Supervised results on the more challenging datasets including SVHN, NORB and CIFAR-10 datasets. Following previous models [34,35,38], we use 1,000 labeled samples on SVHN and NORB datasets to compare our method with other methods. The results show that our method outperforms the previous state-of-the-art.

Inspired by the Grad-CAM [107] on class activation map, we can interpret the classification decision made by our method. We can see that our model is triggered by different semantic regions of the image for different classes of classification. Fig. 3 shows that our GSSL method by using Grad-CAM method provides "visual explanations" for decisions from the all classes of the CNN models. The Fig. 3 indicate the class activation of the model for MNIST and
NORB dataset where we use 100 and 1,000 labeled samples to train the model and use the remaining as test; the result shows the outstanding result of the model in object localization using Grad-CAM technique. We also used T-SNE \cite{108} to visualize the CNN features for training data which are partially supervised. Fig. 2 indicates that the model has acceptable discriminative ability; we applied this method on MNIST and NORB datasets using 100 and 1,000 labeled samples in the training step. The figure shows that our model can discriminate the training data in the embedded space using partially supervised samples.
Chapter 4

A Weakly Supervised Fine Label Classifier Enhanced by Coarse Supervision

Objects are usually organized in a hierarchical structure in which each coarse category (e.g., big cat) corresponds to a super class of several fine categories (e.g., cheetah, leopard). The objects grouped within the same coarse category, but in different fine categories, usually share a set of global features; however, these objects have distinctive local properties that characterize them at a fine level. This chapter addresses the challenge of fine image classification in a weakly supervised fashion, whereby a subset of images is tagged by fine labels (i.e., fine images), while the remaining are tagged by coarse labels (i.e., coarse images). We propose a new deep model that leverages coarse images to improve the classification performance of fine images within the coarse category. Our model is an end-to-end framework consisting of a Convolutional Neural Network (CNN) which uses fine and coarse images to tune its parameters. The CNN outputs are then fanned out into two separate branches such that the first branch uses a supervised low rank self-expressive layer to project the CNN outputs to the low rank subspaces to capture the global structures for the coarse classification, while the other branch uses a supervised sparse self-expressive layer to project them to the sparse subspaces to capture the local structures for the fine classification. Our deep model uses coarse images in conjunction with fine images to jointly explore the low rank and sparse subspaces by sharing
the network parameters during the training which causes the data obtained by the CNN to be well-projected to both sparse and low rank subspaces for classification.

### 4.1 Introduction

Over the past few years, CNNs have provided promising results in object recognition and other visual classification tasks [9, 109, 110]. Along with these developments, image sub-categorization has been used to increase performance of a wide variety of applications in computer vision, such as face recognition [111] and object detection [112]. However, training a CNN requires a vast amount of accurately annotated images [9]. Moreover, providing a sufficient amount of labeled images to train a CNN is labor intensive, time consuming, and usually requires expert knowledge to annotate them accurately, especially where the class of objects is too fine-grained [64, 113].

Objects in the fine classes which are grouped in the same coarse category usually share a set of common visual features. These shared visual properties are typically the global...
structure underlying the objects which allow them to be categorized at a coarse level. However, these objects have distinctive local properties that are used to characterize them at a fine level. For example, consider the images in Fig. 6.1. These images are different species of big cats (e.g., cheetah, jaguar, leopard). We can see that all these animals have spots; this is a global feature that is common among them (i.e., commonalities). However, these animals have distinctive feline features (e.g., cheetahs have a ”tear line” on their faces that leopards do not) which are specific to each species. These are the local features that are disjoint from the common features among these animals which are used for fine image classification.

The low rank and sparse representation of the high dimensional data is based on the fact that a correlation often exists among the data which belong to the same class such that a low rank subspace captures the global and smooth structures, while a sparse subspace captures the local structures and fine details underlying the data [70,114–116].

In this work, we make a structural assumption about all the data points which are extracted from the CNN in our framework. This assumption is based on the fact that there is correlation among the data points which belong to the same class. This structural assumption causes all the data extracted from the CNN to exist in a union of low rank, or approximately low rank, subspaces. These subspaces in our model are explored by using a self-expressive property such that each data point can be represented as a linear combination of other samples in the same subspace. In addition to the low rankness of the data (i.e., low rank self-expressive), we also put one sparsity constraint on the number of data which are used to express a given image from a fine category (i.e., sparse self-expressive). This is because the differences between images in a fine category are very small, so we want any given data point from a fine category to be expressed as a linear combination of a very small number of similar data points. We use low rank and sparse subspaces to represent the coarse and fine concepts of the data to address the challenge of fine and coarse image classification. In our model, sparse and low rank subspaces are explored jointly by sharing the network parameters to take advantage of both the coarse and fine images during the training which causes the data extracted from the CNN to be well-projected to sparse and low rank subspaces for classification [117].
4.2 Supervised Self-Expressive Layers

Before explaining our entire framework in Section 5, in this section we describe our supervised low rank and sparse self-expressive layers in our framework, as shown in Fig. 4.2. These layers incorporate labels to explore sparse and low rank subspaces underlying the data. In this framework, the coarse properties of the data are represented by low rank subspaces, while the fine properties of the data are represented by sparse subspaces. Therefore, the supervised self-expressive layers in our framework have two main characteristics: 1) the low rank self-expressive layer learns the global and coarse concepts of the data, while the sparse self-expressive layer learns the local and fine concepts of the data, and 2) these layers incorporate the label information to increase the class separability by leveraging a contrastive loss function during the exploration of the subspaces.

Note that in the self-expressive layers, coarse concepts can be represented by samples from the fine categories. However, fine concepts may not be represented by coarse samples. For example, consider the coarse category of big cats including cheetah, leopard and jaguar subcategories. When we want to represent the "spots" which is a coarse and common concept among all of the samples from the big cats category, we use samples from all of the fine categories and other samples which are only labeled by big cat as all have this property. However, if we want to represent the "tear line on the face", which is a fine concept, we use the samples from the cheetah class, not samples from the leopard or jaguar classes, since the cheetahs have this property while leopards and jaguar do not. For this reason, we separate the CNN outputs into two different branches, where one of them is used for classifying the coarse concepts and the other one is used for classifying the fine concepts.

4.2.1 Supervised Low Rank Self-Expressive Layer

The main goal of the low rank self-expressive layer in our framework is to explore the subspaces that represent the global and coarse structures of the CNN outputs. Assume that $g(w_s, x_i)$ indicates the feature vector obtained by our CNN for sample $x_i$ (i.e., output of fully connected layer in Fig. 4.2 for sample $x_i$), where $w_s$ is parameters of the CNN. Let $G(w_s, X)$ represent the feature matrix constructed by stacking $[g(w_s, x_1), g(w_s, x_2), ..., g(w_s, x_n)]$ column-
wise. The cost function $SLR(w_s, C_L, X)$, used to train the supervised low rank self-expressive layer is as follows:

$$
||C_L||_* + \frac{\lambda}{2}||G(w_s, X) - G(w_s, X)C_L||_F^2 + \\
\sum_{i=1}^{n} \sum_{j=1}^{n} (1 - y_{ij})||g(w_s, x_i) - g(w_s, x_j)||^2 + \\
y_{ij} \max\{0, (m - ||g(w_s, x_i) - g(w_s, x_j)||)\}^2,
$$

(4.1)

where, $C_L$ is a low rank affinity matrix. The first two terms in (5) are the loss terms which force $G(w_s, X)$ to be projected into the low rank subspaces. In other words, $C_L$ regularizes $w_s$ to learn the low rank structures of features. This is because, based on the rank inequality of multiplying two matrices (i.e., $\{\text{rank}(AB) < \min\{\text{rank}(A), \text{rank}(B)\}\}$), we can conclude that, by minimizing (5) with respect to $w_s$, $G(w_s, X)$ is projected into subspaces such that $\text{rank}(G(w_s, X)) < \text{rank}(C_L)$. Note that we simultaneously minimize $\text{rank}(C_L)$ by using the term $||C_L||_*$ in (5) which causes to be reduced the rank of $G(w_s, X)$.

The third term in (5) is a contrastive loss, the goal of which is to bring $g(w_s, x_i)$ and $g(w_s, x_j)$ close to each other if $x_i$ and $x_j$ samples belong to the same coarse category, while pushing them away from each other if they belong to the different coarse categories. $y_{ij} = 0$

Figure 4.2: indicates our deep framework, the CNN output is fanned out into two separate branches for fine and coarse image classification.
if \( x_i \) and \( x_j \) have the same coarse label, otherwise, \( y_{ij} = 1 \). Here, \( m \) is the margin used in the contrastive loss.

### 4.2.2 Supervised Sparse Self-Expressive Layer

The goal of the sparse self-expressive layer in our framework is to explore the subspaces which represent the local and fine structures of the CNN outputs. In the sparse self-expressive layer, we put a sparsity constraint on the number of similar data points which are used to express a given sample \( (x) \) from a fine category. The sparsity constraint prevents the samples \( (Z) \) within the same coarse category, but belonging to the different fine categories, from contributing to the expression of sample \( x \). This is because we want to express the fine concept of \( x \), and \( Z \) may not have the same fine concept. Indeed, we choose the sparse subspace as a solution to express the fine concepts because the differences between images in a fine category are very small, so we want any given data point from a fine category to be expressed as a linear combination of a very small number of similar data points. Eq. (6) shows the loss used to learn the sparse self-expressive layer. The first and second terms in (6) are the loss terms used to explore sparse subspaces, while the third term is the supervised loss used to increase the separability of the fine classes. Minimizing (6) with respect to \( w_s \) forces the feature matrix \( G(w_s, X) \) to be projected in subspaces such that each sample \( g(w_s, x_i) \) can be expressed by the minimum possible number of other samples from the same subspace. This is obtained by putting a constraint on the affinity matrix \( C_S \) so that it has a small \( \ell_1 \) norm. \( y'_{ij} = 0 \) if \( x_i \) and \( x_j \) belong to the same fine category, otherwise \( y'_{ij} = 1 \). Our supervised sparse self-expressive layer cost function, \( SS(w_s, C_S, X) \), is defined as follows:

\[
||C_S||_1 + \frac{\lambda}{2} ||G(w_s, X) - G(w_s, X)C_S||_F^2 + \\
\sum_{i=1}^{n} \sum_{j=1}^{n} (1 - y'_{ij}) ||g(w_s, x_i) - g(w_s, x_j)||^2 + \\
y'_{ij} \max\{0, (m - ||g(w_s, x_i) - g(w_s, x_j)||)\}^2,
\]

where, \( C_S \) is a sparse affinity matrix. Note that in the sparse subspaces of our model, the structural assumption (i.e., low rankness) among the data points is preserved as well. This is
because the two branches of the framework are trained jointly by sharing the parameters \((w_s)\) during the training such that the rank of \(G(w_s, X)\) is attempted to be minimized in the low rank self-expressive layer.

### 4.3 Fine-Coarse Label Classifier Framework

Fig. 4.2 illustrates our complete architecture. This architecture is an end-to-end framework consisting of a CNN whose output is fanned out into two separate branches. The first branch projects the CNN outputs into the low rank subspaces by using a supervised low rank self-expressive layer. The data points projected into the low rank subspaces are then classified into the coarse categories by using a softmax layer \((w_c)\) in Fig. 4.2. The second branch, however, projects the CNN outputs into the sparse subspaces by using a supervised sparse self-expressive layer. The data points projected into the sparse subspaces are then classified into the fine classes by using a softmax layer \((w_f)\) in Fig. 4.2.

Eq. (7) indicates the total loss function that our deep model uses for joint fine and coarse image classification. In this loss function, both the fine and coarse classification tasks share the parameters \((w_s)\) to jointly explore the sparse and low rank subspaces by optimizing the Eq. (5) and Eq. (6) simultaneously during the training phase. In this framework, coarse images contribute to the fine classification because images with the same coarse label, but belonging to different fine categories, still have a common global structure that can be used in conjunction with fine images to better tune the parameters of our model \((w_s)\) during the training phase. In such a case, the coarse images affect the parameters such that the CNN outputs are well-projected into both low rank and sparse subspaces which contributes to the fine classification. The parameters which are optimized in our architecture are \(\{w_c, w_f, w_s, C_L, C_S\}\). The total loss of the framework, \(\mathcal{L}(.)\), is formulated as follows:

\[
\mathcal{L}(w_s, w_c, w_f, C_L, C_S) = \\
\phi(SS(w_s, C_S, X) + \gamma(SLR(w_s, C_L, X)) + \\
\mathcal{L}_c(w_cG(w_s, X), L_c) + \mathcal{L}_c(w_fG(w_s, X), L_f),
\]

(4.3)

where, \(\mathcal{L}_c(.)\) is the softmax cross entropy loss function used for classification. The \(SLR\) and \(SS\)
are the supervised sparse and low rank self-expressive layer loss functions defined in (5) and (6), respectively. $L_c$ and $L_f$ are the coarse and fine ground truth labels, respectively. $w_c G(w_s, X)$ and $w_f G(w_s, X)$ are the predicted coarse and fine labels by our model, respectively. $\phi$ and $\gamma$ are the hyper-parameters that balance different loss terms during the training.

### 4.4 Experiments and Implementation Details

#### 4.4.1 CNN Architecture

We use a VGG [110] architecture as shown in Fig. 4.2. We apply batch normalization [118] after each convolutional layer, and before performing the Rectified Linear Units (ReLU) activation function [9]. We use an Adam optimizer [105] with the default hyper-parameters values ($\epsilon = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$) to train the parameters of the CNN. The batch size in all experiments is fixed to 128 and the framework is implemented in TensorFlow.

#### 4.4.2 Training the Framework

We arrange a strategy which has two steps to train the parameters of the entire network: pre-training and fine-tuning. This strategy also prevents the trivial all-zero solution while minimizing the losses defined in (5) and (6). In the pre-training step, each component of the framework (i.e., CNN network ($w_s$), first branch ($w_c, C_L$) and second branch ($w_f, C_S$) is trained separately, while in the fine-tuning step, all of the components of the framework are trained jointly.
Pre-training Step

In the pre-training step, we first initialize the CNN parameters ($w_s$) by a VGG-Net pre-trained on a subset of the ImageNet 2010 [64]. In the next step of the pre-training step, we train the parameters of each branch separately. In each branch, there are two tasks that are learned successively: 1) projecting the CNN outputs (i.e., $G(w_s,X)$) into the low rank and sparse subspaces, and 2) classifying the projected data points into the coarse and fine categories. Therefore, in the first branch, the first task updates $C_L$, and the second task updates the coarse classifier parameters ($w_c$). In the second branch, however, the first task updates $C_S$, and the second task updates the fine classifier parameters ($w_f$).

Note that $C_L$ in the first branch and $C_S$ in the second branch can be thought of as the parameters of an additional network layer (i.e., self-expressive layer as it is shown in Fig. 4.2), which allows us to find a solution for $C_L$ and $C_S$ in (5) and (6) by using back-propagation in the first and second branch, respectively. We also note that the self-expressive layers in our framework are not a fully connected layer. In the low rank-self-expressive layer, each data point ($z_i = g(w_s,x_i)$ in Fig. 4.2 where $n$ is the number of training data) is only connected to all the data which belong to the same coarse category. Thus, data within the same coarse category construct a complete bipartite graph in the low rank-self-expressive layer. In the sparse-self-expressive layer, however, each data point attempts to be connected to only the minimum possible number of data points which belong to the same fine category. Thus, in this layer, data points within the same fine category construct a bipartite graph which is very sparse.

Fine-tuning Step

In the fine-tuning step, all parameters of the framework are updated jointly. The parameters of the model in this step have been initialized with a proper starting point obtained from the pre-training step. In this step, we aim to minimize the total loss function in (7), all at once. Since the framework is trained in batch mode through several epochs, we optimize (7) with respect to $C_L$ and $C_S$ parameters only after each epoch is finished, where we have observed all the training data, because $C_L$ and $C_S$ need to access all the data
points when they project the data points into the subspaces with the self-expressive property. Therefore, we train the framework in batch mode and optimize (7) with respect to all the parameters excluding $C_L$ and $C_S$. Then, at the end of each epoch, we minimize the total loss with respect to $C_L$ and $C_S$ (i.e., this step is similar to the alternative minimization algorithm, where we optimize one parameter alternatively by fixing all other parameters). Note that during the training, we store the CNN features of the images. This process avoids the need to re-calculate those features (i.e., $G(w_s, X)$), when we update $C_L$ and $C_S$ parameters.

4.4.3 Dataset and Experimental Setup

We arrange our experimental setup as described in [64] and [66]. We perform our experiments on a subset of the ImageNet 2010. This subset consists of the classes from the ImageNet 2010 which have only one parent class. Among these classes, the parent classes form the coarse categories, $S_{coarse}$, while their corresponding children contain the fine categories, $S_{fine}$. Using this setup, we have $|S_{coarse}| = 143$ coarse classes, and $|S_{fine}| = 387$ fine classes. Since in this work, we consider fine image classification in a weakly supervised fashion, the original training, validation, and test sets of the ImageNet 2010 dataset are truncated to $S_{fine}$. The truncated training set contains 487K images in which there are between 1.4K and 9.8K images for each coarse category, and between 668 and 2.4K images for each fine category. For the validation and the test sets, there are 50 and 150 images per fine category, respectively. In all of our experiments, we report the performance of the model by using the top-one average accuracy, as reported in [64] and [66]. We randomly divide the truncated training set, $S$, into two disjoint subsets for each fine category. In this case, the first subset of images, $S_{coarse}$, have only the coarse labels, while the second subset of images, $S_{fine}$, have fine labels as well.

4.4.4 Hyper-parameters Tuning

We adjust the hyper-parameters of the model based on a range of the values that provide the best accuracy on the validation set. We set $|S_{coarse}| = |S_{fine}| = 0.5|S|$ in the training set during the tuning of the hyper-parameters. The hyper-parameters of our model are
{γ, φ, λ, m} defined in (7) in which m and λ are used in the \( SS(.) \) and \( SLR(.) \) functions. Fig. 4.3 indicates the accuracy of the model on the validation set in tuning the hyper-parameters. Fig. 4.3(a) indicates the performance of the model by setting \( λ = \{1, 10, 100, 500, 1000\} \), the results indicate that \( λ = 100 \) provides the best accuracy. When choosing \( m \) to be \( \{0.2, 0.5, 1, 1.5\} \), as shown in Fig. 4.3(b) the best value for the margin, \( m \), is 1. When setting \( γ \) to be \( \{0.01, 0.1, 1, 5, 10, 100\} \) and \( φ \) to be \( \{1, 5, 10, 100\} \), as shown in Fig. 4.3(c) and Fig. 4.3(d), values of \( γ = 1 \) and \( φ = 10 \) provide the best accuracy.

### 4.4.5 Evaluating the Model in Different Cases

In this section, we evaluate performance of our model—Deep Fine Classifier (DFC), in five different cases. DFC-S is the case where we train each component of the framework 'separately' (pre-training step defined in the Section 6.2.1). However, in this case we ignore the contrastive loss in (5) and (6) during exploring the subspaces. DFC-S-CL is the case where we add contrastive loss to the DFC-S case.

DFC-J is the case where we train the entire framework 'jointly' by using the loss (7), but ignore the contrastive loss in (5) and (6). DFC-J-CL is the case where we add contrastive loss to the DFC-J case, in which we train the entire framework 'jointly' by considering the contrastive loss during the exploration of subspaces in (5) and (6). Finally, Base-J-CL is the case where we remove the sparse and low rank constraints (i.e., we remove the first two terms from (5) and (6)) and then train the network using the loss (7).
Methods & $|S| = 0.1|S|$ & $|S| = 0.2|S|$ & $|S| = 0.5|S|$ \\
Base-J-CL & 69.19 & 70.51 & 72.64 \\
DFC-S & 72.84 & 74.46 & 77.58 \\
DFC-S-CL & 74.32 & 76.16 & 79.41 \\
DFC-J & 76.23 & 77.39 & 80.12 \\
DFC-J-CL & 76.89 & 78.83 & 81.17 \\

Table 4.1: Comparing performance of our model in different scenarios, we fix $|S_{coarse}| = 0.5|S|$, and we set $|S_{fine}|$ to be $0.1|S|$, $0.2|S|$ and $0.5|S|$. 

**Supervised Sparse and Low Rank Layers**

In a supervised version of exploring low rank and sparse subspaces, we incorporated the label by using a contrastive loss function. To see the effectiveness of this process in our model, we compare DFC on the testing set for two pairs of different cases (i.e., $\{DFC-S, DFC-S-CL\}$, and $\{DFC-J, DFC-J-CL\}$). Table 4.1 indicates the performance of the model in different cases on the testing set where we fix $|S_{coarse}| = 0.5|S|$ and set $|S_{fine}|$ to be $0.1|S|$, $0.2|S|$ and $0.5|S|$, respectively. Table 4.2 shows the accuracy of the model on the testing set where we fix $|S_{fine}| = 0.5|S|$ and set $|S_{coarse}|$ to be $0.1|S|$, $0.2|S|$ and $0.5|S|$, respectively.

By comparing DFC-J with DFC-J-CL in Table 4.1 and Table 4.2, we observe that including contrastive loss during the exploration of the subspaces, in either configuration of splitting training data to the coarse and fine labels, increases our model performance. We can see that fine label classifier performance in our model is improved by 1.05% on average (Table 4.1) when we fix $|S_{coarse}| = 0.5|S|$ and vary the size of $|S_{fine}|$, while it is improved by 0.83% on average (Table 4.2) when we fix $|S_{fine}| = 0.5|S|$ and vary the size of $|S_{coarse}|$. Moreover, by comparing DFC-S with DFC-S-CL, the average improvement of the model in Table 4.1 and Table 4.2 are 1.67% and 1.20%, respectively. Finally, the comparison of results between Base-J-CL and DFC-J-CL in Table 4.1 and Table 4.2 shows that the LR and sparse constraints have a significant impact on our model.

**Effectiveness of Joint Training**

In this section, we show the effectiveness of joint-training for our framework. The results reported in Table 4.1 and Table 4.2 show that training the model jointly (i.e., DFC-J-CL case) significantly improves the overall performance of the framework in comparison to
Table 4.2: Comparing performance of our model in different scenarios, we fix $|S_{\text{fine}}| = 0.5|S|$, and we set $|S_{\text{coarse}}|$ to be 0.1$|S|$, 0.2$|S|$ and 0.5$|S|$.

| Size         | 0.1$|S|$ | 0.2$|S|$ | 0.5$|S|$ |
|--------------|--------|--------|--------|
| DFC-J-CL-Base| 73.92 ± 0.16 | 76.31 ± 0.21 | 79.06 ± 0.14 |
| DFC-J-CL     | 76.89 ± 0.24 | 78.83 ± 0.18 | 81.17 ± 0.11 |

Table 4.3: The model accuracy by using and not using coarse data.

the case where we train our model components separately (i.e., DFC-S-CL case). These improvements, on average, are 2.33% and 1.82% in Table 4.1 and Table 4.2, respectively.

### 4.4.6 Impact of Fine and Coarse Samples in Training

Here we investigate the effect of fine images during the training. Moreover, we demonstrate how coarse images improve the model performance by fixing the fine image set size during the training. Table 4.1 shows the case in which we fix the coarse image set size (i.e., $|S_{\text{coarse}}| = 0.5|S|$) while we increase the fine image set size ($|S_{\text{fine}}|$) during the training. As shown in Table 4.1, the overall performance of our entire framework (i.e., DFC-J-CL) increases by 1.94% and 2.34% as we increase the size of $|S_{\text{fine}}|$ from 0.1$|S| \rightarrow 0.2|S|$ and 0.2$|S| \rightarrow 0.5|S|$, respectively. Table 4.2 shows the effectiveness of the incorporation of coarse images into our model. Table 4.2 shows the case where we fix the fine images set size (i.e., $|S_{\text{fine}}| = 0.5|S|$) while we increase the coarse image set size ($|S_{\text{coarse}}|$) during the training. As shown in Table 4.2, the overall performance of our model (i.e., DFC-J-CL) increases by 0.69% and 1.33% as we increase the size of $|S_{\text{coarse}}|$ from 0.1$|S| \rightarrow 0.2|S|$ and 0.2$|S| \rightarrow 0.5|S|$, respectively.

We also evaluated our entire model (i.e., DFC-J-CL) for a naive baseline called (DFC-J-CL-Base) where we use two branches, one for fine and the other one for coarse classification. However, in this case during the training, we ignore coarse images which are not tagged by fine labels and we only use images which have both the fine and coarse labels. We set
Table 4.4: Comparing our method with others by fixing $|S_{\text{coarse}}| = 0.5|S|$, and setting $|S_{\text{fine}}|$ to be $0.1|S|$, $0.2|S|$ and $0.5|S|$, and ran this baseline. By comparing this baseline with DFC-J-CL in Table. 4.3, we observe that our model improves fine classification on average by 2.97 %, 2.52 %, and 2.11 % when we use coarse labeled images with missing fine labels. Note that the DFC-J-CL-Base and DFC-J-CL will be the same case if we use 100% of images with fine and coarse labels and the accuracy of our model in this case is 85.67 ± 0.26%.

4.4.7 Comparison

We compare our deep fine label classifier with NN-H-RNCMF [64] and its baseline, RNCMF, and RNN [66], which also attempt to enhance fine classification by leveraging the coarse images in hierarchical structure. Moreover, we compare our model with DMSC [119] as a baseline, where a similar sparse and low-rank approach was employed for a clustering task. For this method, we modified it to our task. Specifically, the encoder in this model takes unimodal data as the input data in our case are RGB images. All of the hyper-parameters in these methods are chosen based on the authors suggestion. All of these methods use the features extracted from the VGGNet pre-trained on the subset of ImageNet dataset [64]. We are consistent with these methods regarding the amount of coarsely and finely labeled images during the training. We set $|S_{\text{coarse}}| = 0.5|S|$, and change $|S_{\text{fine}}|$ to $\{0.1|S|, 0.2|S|, 0.5|S|\}$. The results in Table. 4.4 show that DFC-J-CL outperforms other methods in all configurations, indicating its great potential to leverage coarse images for improving fine image classification. In further study, we compared our method in a case where all the fine images are available during the training. In this configuration, our model accuracy is 85.67%, while the accuracy of the methods including the baseline VGG fine-tuned on the ImageNet2010, NN-H-RNCMF, and RNN are 76.01%, 74.18%, and 82%, respectively as reported in [64,66].
4.4.8 Further Analysis

Inspired by [120], we use a Grad-CAM to expose the implicit attention of our model on the images during the classification. We observe that our model is triggered by semantic regions of the images for fine image classification. By using the Grad-CAM, we can see that our fine label classifier provides a "visual explanation" for the decision that it makes during the classification. Fig. 4.6 indicates the Grad-CAM in our model for 'bear', 'fox', 'wolf' and 'spider' species. It shows that our model makes decisions by using the appropriate regions of the images to classify these species.

Furthermore, we used T-SNE [108] to visualize the CNN features for training in two cases where we train the model separately and jointly. Fig. 4.4(a) illustrates the case where we train our model separately, while Fig. 4.4(b) shows the case where we train the model jointly. Fig. 4.4(a) and Fig. 4.4(b) shows that the training data in joint-training are better separated than separate-training case.

Moreover, we visualized the affinity matrices of sparse and low rank subspaces for coarse categories. We selected 100 samples of five classes including 'bear', 'fox', 'wolf', 'spider' and 'grouse' with the same coarse labels. Fig. 4.4(c) shows the sparse affinity matrix, $C_S$, and Fig. 4.4(d) indicates the low rank affinity matrix, $C_L$, for the coarse categories. Fig. 4.4(c) shows that affinity matrix, $C_S$, is much sparser than affinity matrix, $C_L$. This means that the fine data points are expressed by few number of samples, while the coarse data are expressed by more samples in the same subspace.

In further study, we visualized the affinity matrices of sparse and low rank subspaces for fine categories. We picked 100 samples from the 'fox' category by choosing 25 samples
Figure 4.6: Example of Grad-CAM for classifying different species.

per each fine category including ‘artic’, ‘grey’, ‘kit’, and ‘red’. Fig. 4.5(a) shows the sparse affinity matrix, $C_S$, and Fig. 4.5(b) indicates the low rank affinity matrix, $C_L$, for the fine categories. Fig. 4.5(b) shows that almost all samples contribute to express a given sample in the low rank subspace, while Fig. 4.5(a) shows that these samples are grouped to four categories such that each sample can be expressed by the other samples from the same group.
Chapter 5

Hierarchical Optimal Transport for Semi-Supervised Learning

Semi-Supervised Learning (SSL) based on Convolutional Neural Networks (CNNs) have recently been proven as powerful tools for standard tasks such as image classification when there is not a sufficient amount of labeled data available during the training. In this work, we consider the general setting of the SSL problem for image classification, where the labeled and unlabeled data come from the same underlying distribution. We propose a new SSL method that adopts a hierarchical Optimal Transport (OT) technique to find a mapping from empirical unlabeled measures to corresponding labeled measures by leveraging the minimum amount of transportation cost in the label space. Based on this mapping, pseudo-labels for the unlabeled data are inferred, which are then used along with the labeled data for training the CNN. We evaluated and compared our method with state-of-the-art SSL approaches on standard datasets to demonstrate the superiority of our SSL method.

5.1 Introduction

Training a CNN model relies on large annotated datasets, which are usually tedious and labor intensive to collect [17]. Two approaches are usually considered to address this problem: Transfer Learning (TL) and Semi-Supervised Learning (SSL). In TL [23], learning of a new task is improved by transferring knowledge from a related task which has
already been learned. However, in SSL [24], learning of a new task is improved by using information from an input distribution that is provided by a large amount of unlabeled data. To make use of the unlabeled data, it is assumed that the underlying distribution of this data follows at least one of the following structural assumptions: continuity, clustering, or manifold [16]. In the continuity assumption [25–27], data points close to each other are more likely to belong to the same class. In the clustering assumption [25, 28, 29], data tend to form discrete clusters, and data in the same cluster are more likely to share the same label. In the manifold assumption [30, 31], data lie approximately on a manifold of much lower dimension than the input space which can be classified by distances between probability measures on the manifold [32]. To quantify the difference between two probability measures on a manifold properly, modeling the geometrical structures of the manifold is required [121–123].

One of the methodologies used to model geometrical structures on the probability simplex (i.e., manifold of discrete probability measures) is grounded on the theory of Optimal Transport (OT) [123, 124]. The Wasserstein distance, which arises from the idea of OT, exploits prior geometric knowledge of the base space in which random variables are valued [123]. Computing the Wasserstein distance between two random variables amounts to achieving a transportation plan which requires the minimal expected cost. The Wasserstein distance considers the metric properties of the base space in which a pattern is defined [121]. This characteristic of the Wasserstein distances in OT has attracted a lot of attention for machine learning and computer vision tasks such as computing the barycenters [125, 126] of multiple distributions [127], generating data [52], designing loss function [128], domain adaptation [129–134], and clustering [135–138].

Data are usually organized in a hierarchical structure, or taxonomy. For example, considering a set of data belonging to the same class in a dataset as a measure, we can think of all the data in the dataset as a measure of measures. Inspired by OT, which maps two measures with the minimum amount of transportation cost, we can think of using hierarchical OT to map two measure of measures such that the total transportation cost across the measures becomes minimum. In future work, we propose an SSL method that leverages from hierarchical OT to map measures from an unlabeled set to measures in a labeled set with a minimum amount of the total transportation cost in the label space.
Our method stems from two basic premises: 1) Data in a given class in the labeled and unlabeled sets come from the same distribution. 2) Assume we are given three measures with roughly the same amount of data, where only two of these measures come from the same distribution. The OT cost between two measures from the same distribution is expected to be less than the OT cost between one of these measures and the measure from a different distribution. Following these premises, we thus expect that the hierarchical OT maps measures from the same distribution in the labeled and unlabeled sets such that the total transportation cost between two measure of measures becomes minimum. Based on this mapping, a pseudo-label for unlabeled data in each measure from the unlabeled set is inferred. These unlabeled data annotated by pseudo-labels are then used along with the labeled data to train a CNN. However, data in the unlabeled set are not labeled to allow us to identify the measures. Thus, following the clustering assumption in SSL and the role of OT in clustering [135–138], we can consider all the measures in the unlabeled set as a group of clusters which are identified by the Wasserstein barycenters of the unlabeled data [139].

5.2 Preliminaries

5.2.1 Discrete OT and Dual Form

For any $r \geq 1$, let the probability simplex be denoted by $\Delta_r = \{ q \in \mathbb{R}^r : q_i \geq 0, \sum_{i=1}^{r} q_i = 1 \}$, and also assume that $X = \{ x_1, ..., x_n \}$ and $X' = \{ x'_1, ..., x'_m \}$ are two sets of data points in $\mathbb{R}^d$ such that $\mathcal{X} = \sum_{i=1}^{n} a_i \delta_{x_i}$ and $\mathcal{X}' = \sum_{i=1}^{m} b_i \delta_{x'_i}$ in which $\delta_{x_i}$ is a Dirac unit mass located on point $x_i$, and $a, b$ are the weighting vectors which belong to the probability simplex $\Delta_n$ and $\Delta_m$, respectively. Then, the Wasserstein-$p$ distance $W_p(\mathcal{X}, \mathcal{X}')$ between two discrete measures $\mathcal{X}$ and $\mathcal{X}'$ is the $p$-th root of the optimum of a network flow problem known as the transportation problem [140]. The transportation problem depends on two components: 1) matrix $M \in \mathbb{R}^{n \times m}$ which encodes the geometry of the data points by measuring the pairwise distance between elements in $X$ and $X'$ raised to the power $p$, 2) the transportation polytope $\pi(a, b) \in \mathbb{R}^{n \times m}$ which acts as a feasible set, characterized as a set of $n \times m$ non-negative matrices such that their row and column marginals are $a$ and $b$, respectively. This means
that the transportation plan should satisfy the marginal constraints. In other words, let $1_m$ be an $m$-dimensional vector with all elements equal to one, then the transportation polytope is represented as follows: $\pi(a,b) = \{T \in \mathbb{R}^{n \times m} | T^\top 1_n = b, T1_m = a\}$. Essentially, each element $T(i,j)$ indicates the amount of mass which is transported from $i$ to $j$. Note that in the transportation problem, the matrix $M$ is also considered as a cost parameter such that $M(i,j) = d(x_i, x'_j)^p$ where $d(.)$ is the Euclidean distance.

Let $\langle T, M \rangle$ denote the Frobenius dot-product between $T$ and $M$ matrices. Then, the discrete Wasserstein distance $W_p(\mathcal{X}, \mathcal{X}')$ is formulated by an optimum of a parametric linear program $g(.)$ on a cost matrix $M$, and $n \times m$ number of variables parameterized by the marginals $a$ and $b$ as follows:

$$W_p(\mathcal{X}, \mathcal{X}') = g(a, b, M) = \min_{T \in \pi(a,b)} \langle T, M \rangle. \quad (5.1)$$

The Wasserstein distance in (1) is a Linear Program (LP) and a subgradient of its solution can be calculated by Lagrange duality. The dual LP of (1) is:

$$d(a, b, M) = \max_{(\alpha, \beta) \in C_M} \alpha^\top a + \beta^\top b, \quad (5.2)$$

where the polyhedron $C_M$ of dual variables is as follows:

$$C_M = \{(\alpha, \beta) \in \mathbb{R}^{m+n} | \alpha_i + \beta_j \leq M(i,j)\}. \quad (5.3)$$

Considering LP duality, the following equality is established: $d(a, b, M) = p(a, b, M) \quad (140)$. Computing the exact Wasserstein distance is time consuming. To alleviate this, in $[141]$, Cuturi has introduced an interesting method that regularizes (1) using the entropy of the solution matrix, $H(T)$, (i.e., $\min\langle T, M \rangle + \gamma H(T)$ , where $\gamma$ is regularization strength). It has been shown that if $T'_\gamma$ is the solution of the regularized version of (1) and $\alpha'_\gamma$ is its dual solution in (2), then $\exists u \in \mathbb{R}^n, v \in \mathbb{R}^m$ such that the solution matrix is $T'_\gamma = \text{diag}(u)K\text{diag}(v)$ and $\alpha'_\gamma = -\log(u)/\gamma + (\log(u)^\top 1_n)/(\gamma n))1_n$ where, $K = \exp(-M/\gamma)$. The vectors $u$ and $v$ are updated iteratively between step 1 and 2 by using the well-known Sinkhorn algorithm as follows: step 1 : $u = a/Kv$ and step 2 : $v = b/K^\top u \quad [141]$. 
5.2.2 Hierarchical OT

Let $\theta$ be a Polish space and $S(\theta)$ be the space of Borel probability measures on $\theta$. Since $\theta$ is a Polish space, $S(\theta)$ is also Polish space and can be metrized by the Wasserstein distance $[142]$. Considering the recursion of concepts, $S(S(\theta))$ is also a Polish space and is defined as a space of Borel probability measure on $S(\theta)$, which we can then define a Wasserstein distance on this space by using the Wasserstein metric in $S(\theta)$ (Section 3 in [142]). The concept of Wasserstein distance on the measure of measures, $S(S(\theta))$, which is also referred to as Hierarchical OT, is a practical and efficient solution to include structure in the regular OT distance $[143-146]$. Hierarchical OT is used to model the data which are organized in a hierarchical structure, and has been recently studied for tasks such as multimodal distribution alignment $[146]$, document representation $[145]$, multi-level clustering $[137]$ and a similarity measure between two hidden Markov models $[147]$.

Let $\mathcal{D} = \{X_1, X_2, ..., X_n\}$ and $\mathcal{D}' = \{X'_1, X'_2, ..., X'_m\}$ be two sets of measures such that $\mathcal{M} = \sum_{i=1}^{n} r_i \delta X_i$ and $\mathcal{M}' = \sum_{i=1}^{m} s_i \delta X'_i$ in which $\delta X_i$ is a Dirac mass located on the measure $X_i$, and $r$ and $s$ denote the weighting vectors belonging to the probability simplex $\Delta_n$ and $\Delta_m$, respectively. Then, the hierarchical OT distance between $\mathcal{M}$ and $\mathcal{M}'$ can be formulated by a linear program as follows:

$$W_p'(\mathcal{M}', \mathcal{M}) = \min_{T' \in \pi'(r,s)} \sum_{i=1}^{n} \sum_{j=1}^{m} T'(i,j) W_p(X_i, X'_j),$$

(5.4)

where $\pi'(r,s) = \{T' \in \mathbb{R}^{m \times n}|T'^\top 1_m = r, T'^\top 1_n = s\}$, and $W_p(., .)$ is the Wasserstein-$p$ distance between two discrete measures $\mathcal{X}_i$ and $\mathcal{X}'_j$ which is obtained by Eq. (1). In Eq. (4), we have expanded Eq. (1) such that $T'(i,j)$ represents the amount of mass transported from $\delta X_i$ to $\delta X'_j$, and $W_p(., .)$ is the ground metric which has been substituted by the Euclidean distance in Eq. (1) to represent hierarchical nature of the similarity metric between $\mathcal{M}'$ and $\mathcal{M}$. 
Figure 5.1: At each epoch, a small amount of unlabeled data is processed through the current CNN and clustered into $k$ groups. Then, the Wasserstein-2 distance is computed between these groups and the ones formed by the labeled data. Next, a regularized OT is used to form an optimal coupling between the groups from the unlabeled data and the labeled ones, using the Wasserstein-2 distance as cost function (i.e., hierarchical OT). Finally, this coupling provides pseudo-labels for the selected unlabeled data to perform a gradient descent step on the CNN. Here, circles represent unlabeled data and triangles show the labeled data and their color indicate their labels.

5.2.3 Wasserstein Barycenters

Given $N \geq 1$ probability measures with finite second moments $\{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_N\} \in S_2(\theta)$, their Wasserstein barycenters is a minimizer of $F$ over $S_2(\theta)$ where \[125\]:

$$F(\tilde{X}) = \inf_{\tilde{X} \in S_2(\theta)} \frac{1}{N} \sum_{i=1}^{N} W_2^2(\tilde{X}, \mathcal{X}_i). \quad (5.5)$$

In the case where $\{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$ are discrete measures with finite number of elements, each with size $e_i$, the problem of finding Wasserstein barycenters $\tilde{X}$ on the space of $S_2(\theta)$ in (5) is recast to search only on a simpler space $\mathcal{O}_r(\theta)$, where $\mathcal{O}_r(\theta)$ is the set of probability measures with at most $r$ support points in $\theta$, and $r = \sum_{i=1}^{N} e_i - N + 1$ \[148\]. There are fast and efficient algorithms that find local solutions of the Wasserstein barycenters problem over $\mathcal{O}_r(\theta)$ for $r \geq 1$, which the use of these algorithms for clustering has also been studied in \[135,149\].

5.3 Method

Here, we describe our SSL model as is shown in Figure 6.1. Here, data belonging to the same class is defined as a measure. Thus, all the initial labeled data hierarchically are considered as a measure of measures. Similarly, all the unlabeled data are also a measure.
Algorithm 1: Finding Unlabeled Measures via Wasserstein Metric

input: $Q \in \mathbb{R}^{c \times m}$.
1: initialize: $H \in \mathbb{R}^{c \times k}$, $b = \frac{1}{m}$, $t = 1$, $\eta = 0.5$.
2: while $H$ and $a$ have not converged do

3: Maximization Step:
4: set $\hat{a} = \tilde{a} = \frac{1}{m}$.
5: while not converged do
6: $\beta = \frac{(t + 1)}{2}$, $a \leftarrow (1 - \beta^{-1})\hat{a} + \beta^{-1}\tilde{a}$.
7: $\alpha \leftarrow ma' t$: dual optimal form of OT $d(a, b, M_{HQ})$.
8: $\tilde{a} \leftarrow \tilde{a} \circ e^{-\beta a'}; \hat{a} \leftarrow \hat{a}/\hat{a}'^T 1_n$.
9: $\hat{a} \leftarrow (1 - \beta^{-1})\hat{a} + \beta^{-1}\tilde{a}$, $t \leftarrow t + 1$.
10: end while
11: $a \leftarrow \hat{a}$.

12: Expectation Step:
13: $T' \leftarrow$ optimal coupling for $p(a, b, M_{HQ})$.
14: $H \leftarrow (1 - \eta)H + \eta(QT'\tau)\text{diag}(a^{-1})$, $\eta \in [0, 1]$.
15: end while

of measures, each of which is constructed by data belonging to the same class. Following the basic premises mentioned earlier in the introduction, we use a hierarchical OT to predict pseudo-labels for the unlabeled measures to train a CNN. Our method is a three steps iterative algorithm. In the first step, we make a clustering assumption about the unlabeled data and consider all the unlabeled measures as a group of clusters which are identified by the Wasserstein barycenters of the unlabeled data. In the second step, we use the hierarchical OT to map each of the unlabeled measures to a corresponding labeled measure, based on which, a pseudo-label for the data within each of the clusters is predicted. Finally, unlabeled data annotated with pseudo-labels from the second step are used along with the initial labeled data to train the CNN.

5.3.1 Finding Unlabeled Measures via Wasserstein Metric

Given an image $z_i \in \mathbb{R}^{m \times n}$ from either the labeled or unlabeled dataset, CNN acts as a function $f(w, z_i) : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^c$ with the parameters $w$ that maps $z_i$ to a $c$-dimensional output, where $c$ is number of the classes. Assume that $X = \{x_1, ..., x_m\}$ and $X' = \{x'_1, ..., x'_m\}$ are the sets of $c$-dimensional outputs represented by the CNN for the labeled and unlabeled images, respectively. Let $P_i = 1/n_i \sum_{j=1}^{n_i} \delta_{x_j}$ denote a discrete measure constructed by labeled data in the $i$-th class, where $\delta_{x_j}$ is a Dirac unit mass on $x_j$ and $n_i$ is number of the data in the $i$-th class. Then, all of the labeled data form a measure of measures as follows:
\( \mathcal{M} = \sum_{i=1}^{c} \mu_i \delta_{P_i} \), where \( \mu_i = n_i / m \) represents amount of the mass in the measure \( P_i \) and \( \delta_{P_i} \) is a Dirac unit mass on the measure \( P_i \). Similarly, unlabeled data construct a measure of measures \( \mathcal{M}' = \sum_{j=1}^{k} \nu_j \delta_{Q_j} \), where each measure \( Q_i \), is created by unlabeled data belonging to the same class, \( \nu_j = n_j' / m \) is amount of the mass in the measure \( Q_j \), and \( \delta_{Q_j} \) is a Dirac unit mass on \( Q_j \). However, data in the unlabeled set are not labeled to allow us to identify \( Q_j \).

One simple solution to find \( Q_j \), is to use the labels that are directly predicted by the CNN on the unlabeled data. In this case, there is no need to form unlabeled measures, since unlabeled data annotated by the CNN can be used directly for training the CNN. However, CNN as a classifier trained on a limited amount of the labeled data simply mis-classifies these unlabeled data. Thus, there is little option other than unsupervised methods, such as clustering to explore the unlabeled data belonging to the same class. This criterion stems from the structural assumption based on the clustering in SSL, where it is assumed that the data within the same cluster are more likely to share the same label. Inspired by the role of OT in clustering [135–138], we leverage the Wasserstein metric to explore these measures underlying the unlabeled data. Specifically, we use the k-means objective incorporated by a Wasserstein metric loss to find \( Q_j \).

Given \( m \) unlabeled data \( x_1', ..., x_m' \in \Theta \), the k-means clustering as a vector quantization method [150] aims to find a set \( C \) containing at most \( k \) atoms \( c_1, ..., c_k \in \Theta \) such that the following objective is minimized:

\[
J(C) = \inf_{c_1, \ldots, c_k} \frac{1}{m} \sum_{i=1}^{m} ||x_i' - c_j||^2. \tag{5.6}
\]

Let \( Q = \frac{1}{m} \sum_{i=1}^{m} \delta_{x_i'} \) be the empirical measure of data \( x_1', ..., x_m' \), where \( \delta_{x_i'} \) is a Dirac unit mass on \( x_i' \). Then, (6) is equivalent to exploring a discrete probability measure \( \mathcal{H} \) including finite number of support points which minimizes:

\[
F(\mathcal{H}) = \inf_{\mathcal{H} \in \mathcal{O}_k(\Theta)} W_2^2(\mathcal{H}, Q). \tag{5.7}
\]

When \( N = 1 \) in (5), then (7) can also be considered as a Wasserstein barycenters problem.
whose solution is studied in [125, 135, 149]. From this perspective as studied by [135], the
algorithm for finding the Wasserstein barycenters introduces an alternative for the popular
Loyd’s algorithm to find local minimum of the k-means objective, where the maximization
step (i.e., the assignment of the weight of each data point to its closest centroid) is equivalent
to computing $m\alpha'$ in dual form of OT (see Eq. (2)), while the expectation step (i.e., the
re-centering step) is equivalent to updating $H$ using the OT. Algorithm 1 presents clustering
algorithm for exploring the unlabeled measures using the Wasserstein metric.

5.3.2 Mapping Measures via Hierarchical OT for Pseudo-Labeling

We design an OT cost function $f(.)$ to map the measures in $\mathcal{M}' = \sum_{j=1}^{k} \nu_j \delta_{Q_j}$ to the
measures in $\mathcal{M} = \sum_{i=1}^{c} \mu_i \delta_{P_i}$ as follows:

$$f(\mu, \nu, G) = \min_{R \in T(\mu, \nu)} \langle R, G \rangle - \omega H(R), \quad (5.8)$$

where $R$ is the optimal coupling matrix in which $R(i, j)$ is amount of the mass that should
be transported from $Q_i$ to $P_j$ to provide an OT plan between $\mathcal{M}'$ and $\mathcal{M}$. Thus, if the
highest amount of the mass from $Q_i$ is transported to $P_r$ (i.e., $Q_i$ is mapped to $P_r$); the
data belonging to the measure $Q_i$ are annotated by $r$ which is the label of the measure $P_r$.
Variable $G$ is the pairwise similarity matrix between measures within $\mathcal{M}$ and $\mathcal{M}'$ in which
$G(i, j) = W_p(Q_i, P_j)$ is the regularized Wasserstein distance between two clouds of data in
$Q_i$ and $P_j$. Note that the ground metric used for computing $W_p(Q_i, P_j)$ is the Euclidean
distance. Moreover, $\langle R, G \rangle$ is the Frobenius dot-product between $R$ and $G$ matrices, and
$T$ is transportation polytope defined as follows: $T(\mu, \nu) = \{ R \in \mathbb{R}^{c \times c} | R^T 1_c = \nu, R 1_k = \mu \}$. Finally, $H(R)$ is entropy of the optimal coupling matrix $R$ used for regularizing the OT, and $\omega$ is regularization strength in Eq. (8). The optimal solution for the regularized OT in (8) is
obtained by Sinkhorn algorithm.

5.3.3 Training CNN in SSL Fashion

In the third step, we use the generic cross entropy as our discriminative loss function to
train our CNN. Let $\{ z_i \}_{i=1}^{b}$ be training batch annotated by true labels $\{ y_i \}_{i=1}^{b}$, and $\{ z'_i \}_{i=1}^{b}$ be
Algorithm 2: Transporting Labels via Hierarchical OT

**input:** LD: \( Z_l = \{ (z_i, y_i) \}_{i=1}^m \), UD: \( Z_u = \{ z_i' \}_{i=1}^n \), balancing coefficients: \( \xi, \omega, \gamma \), learning rate: \( r \), batch size: \( b \), number of clusters: \( k \).

1. Train CNN parameters initially using the labeled data \( Z_l \).
2. **repeat**
   3. Select \( \{ z_i' \}_{i=1}^m \subset Z_u \), where \( m \ll n \).
   4. Compute \( X = \{ x_l \}_{i=1}^m \), \( X' = \{ x_i' \}_{i=1}^n \): softmax output on \( Z_l \) and \( \{ z_i' \}_{i=1}^m \), resp.
   5. \( \{ Q_1, ..., Q_k \} \leftarrow \) cluster on \( X' \) using Algorithm 1.
   6. \( \{ P_1, ..., P_c \} \leftarrow \) group \( X \) to \( c \) classes.
   7. Compute \( \mu_i, \nu_i \) based on the amount of the mass in \( \{ Q_i \}_{i=1}^k \) and \( \{ P_i \}_{i=1}^c \).
   8. **for** each \( Q_i \) and \( P_j \) **do**
      9. \( G(i, j) \leftarrow W_2(Q_i, P_j) \): using regularized OT.
   10. **end for**
   11. \( R \leftarrow \) optimal coupling for \( f(\mu, \nu, G) \) in Eq. (8): using regularized OT.
   12. \( \{ y_i' \}_{i=1}^m \leftarrow \) pseudo-label each cluster \( Q_i \) based on the highest amount of mass transport toward the labeled measure (i.e., argmax \( R(i, :) \)).
   13. **repeat**
      14. Select a mini-batch: \( \{ z_i, z_i' \}_{i=1}^b \subset (\{ z_i' \}_{i=1}^m \cup Z_l) \).
      15. \( w \leftarrow w - r \nabla_w [L(w)] \), using Eq. (9).
   16. **until** for an epoch
   17. **until** a fixed number of epochs

training batch annotated by pseudo-labels \( \{ y_i' \}_{i=1}^b \), and \( c_i \) denotes barycenter of the cluster that sample \( z_i' \) belongs to. Then, the total loss function \( L(.) \), used to train our CNN in an SSL fashion is as follows:

\[
L(w) = \sum_{i=1}^b L_c(f(w, z_i), y_i) + \xi \left( \sum_{i=1}^b L_c(f(w, z_i'), y_i') + \frac{1}{b} \sum_{i=1}^b ||f(w, z_i') - c_i||^2 \right),
\]  

(5.9)

where \( f(w, z_i) \) is output of CNN for images \( z_i \), and \( L_c(.) \) denotes cross entropy, and \( \xi \) is a balancing hyperparameter. Note that the third term in (9) is the center loss [151] which aims to reduce the distance between the unlabeled data and the barycenters of their corresponding cluster to perform a local consistency regularization [25]. For training, we initially train the CNN using the labeled data as a warm up step, and then use OT to provide pseudo-labels for the unlabeled data to train the CNN along with the labeled data for the next epochs. Specifically, after training the CNN using the labeled data, in each epoch, we randomly select the same amount of labeled data from the pool of unlabeled data to compute their pseudo-labels via hierarchical OT. Then, the CNN is trained in a mini-batch mode. Our overall SSL method is described in Algorithm 2.

**Discussion on Time Complexity:** Algorithm 2 has two main computational parts: 1)
clustering the unlabeled data via Algorithm 1 (i.e. line 5), and 2) mapping the measures via HrOT (i.e. lines 8-11). For part (1): we used \cite{135} and there is an analysis for its Time Complexity (TC) in \cite{149} as follows: Let $c$, $n$, $k$, and $i$ be the number of classes, unlabeled data, barycenters, and iterations in EM for Algorithm 1, resp. Based on the analysis provided in \cite{149}, in our case, $N = 1$ (# of distributions), $d = c$ (dimension) and $\tau = 1$ (adjusting the support points for barycenters every $\tau$ iterations). Thus, TC of part (1) is $O(ink) + O(inck) \approx O(inck)$. For part (2): Since TC of computing the regularized OT distance between two sets of data each with size $m$ is $O(m^2)$ \cite{152}, then we need $O((n/c)(n/k)(ck)) = O(n^2)$ to compute matrix $G$ in lines 8-10, and also we need $O(ck)$ to find $R$ in line 11. Thus, TC of part (2) is $O(ck) + O(n^2) \approx O(n^2)$. By summing part (1) and (2), the total TC for inferring spudo-labels on $n$ data is $O(n^2 + inck)$. Note that $i$ is not large due to smoothing \cite{135}.

5.4 Experiments and Setup

Setup: we conduct our experiments on SVHN \cite{99}, CIFAR-10/100 \cite{101}, and Mini-ImageNet \cite{153} datasets. Mini-ImageNet \cite{153} is subset of ImageNet \cite{154} which consists of 100 classes and 600 images per class. For Mini-ImageNet, we follow the SSL setup in \cite{155}. We use 500 and 100 images per class for the training and testing splits, res. Following the prior works in \cite{26,41,44,155}, we use a ResNet-18 network for the Mini-ImageNet, and a “13-layer” network for the SVHN and CIFAR-10/100 to evaluate our model. We use the typical configuration for SVHN and CIFAR-10/100 \cite{44}, and the same for the Mini-ImageNet, i.e., normalizing images using dataset mean and Standard Deviation (SD) together and then performing data augmentation by random horizontal flips and random 4 pixel translations \cite{44}. For training, we use Adam optimizer \cite{105} with a learning rate of $3 \times 10^{-3}$ and the batch size is set to 128. The stopping criteria for the Sinkhorn algorithm is either maxIter = 10,000 or tolerance = $10^{-8}$, where maxIter is the maximum number of iterations and tolerance is a threshold for the integrated stopping criterion based on the marginal differences. The barycenters in Algorithm 1 are initially set to centroids obtained by k-means.

We follow suggested guidelines in \cite{24} to evaluate our model. 1) We report performance
of a fully-supervised baseline since the purpose of SSL is to significantly improve the fully-supervised baseline. 2) We vary the amount of labeled data when reporting the accuracy of our SSL method since a perfect SSL algorithm should remain effective even with small amount of labeled data. 3) We compare our method with the case where the unlabeled data are labeled by CNN using its own prediction during the training. 4) We study performance of the soft-pseudo-labels which can be generated by our model. 5) We compare the OT-based clustering method described in Algorithm 1 vs k-means to show the effectiveness of Wasserstein metric for finding the unlabeled measures in our model. 7) We conduct ablation studies on the clustering resolution, and also we study effect of the center loss as a local consistency regularizer in our SSL model.

**Hyperparameter Tuning:** Following [24, 26], we use a validation set of 5k images for CIFAR-10/100, and standard validation set of 7k images for SVHN to tune hyperparameters.
of our model. For CIFAR-10 and SVHN, we use 1k labeled data, and for CIFAR-100, we use 4k labeled data. The results shown in Figure 5.2 are the mean and SD of error rate on validation set for CIFAR-10. Similarly, for other datasets, we also tuned the hyperparameters. For SVHN, the values chosen for $\gamma$, $\omega$ and $\xi$ are 0.01, 0.001, and 0.75, resp. For CIFAR-100, the values chosen for $\gamma$, $\omega$ and $\xi$ are 0.001, 0.001, 0.5, res. For Mini-ImageNet, we used the same hyperparameters tuned for CIFAR-100.

5.4.1 Fully Supervised and Deep SSL Methods

We report the error rate of the ”13-layer” CNN on CIFAR-10/100 and SVHN datasets and ResNet-18 on the Mini-ImageNet dataset for both cases where we only use the labeled data (i.e., Supervised in Table. 5.1 & 6.1), and the case where we leverage the unlabeled data using the hierarchical OT technique during the training (i.e., HrOT in Table. 5.1 & 6.1). All of the compared SSL methods in Table. 5.1 & 6.1 use a common CNN architecture. Following the prior works [26, 41, 44, 155], to compare HrOT with other SSL algorithms, we selected the same amount of data in the training set as the labeled data and the remaining as the unlabeled data for SVHN (73k), CIFAR-10/100 (50k) and Mini-ImageNet (50k) datasets. We run our SSL algorithm over 5 times with different random splits of labeled and unlabeled sets for each dataset, and we report the mean and SD of the test error rates. The results in Table. 5.1 & 6.1 indicate the potential of our model for leveraging the unlabeled data in comparison to other SSL methods.

5.4.2 Soft-Pseudo-Labels based on Hierarchical OT

Other than particular manner in HrOT where we choose one particular pseudo-label based on the highest amount of mass transport from an unlabeled measure to a labeled measure to label the unlabeled measure, we also use ”soft pseudo-labels” for training the CNN. In other words, instead of having one-hot target in the usual classification loss, we use the row of the transportation mass corresponding to the labeled measures as the target. The compared result in Table. 5.1 & 6.1 show that using one-hot targets (HrOT) outperforms using soft pseudo-labels (Soft-HrOT). The reason can be supported by SSL methods based
Table 5.2: Comparing test error between HrOT and different baselines and SSL methods.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>CIFAR-100</th>
<th>Mini-ImageNet-top1</th>
<th>Mini-ImageNet-top5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4000</td>
<td>10000</td>
<td>4000</td>
</tr>
<tr>
<td>Supervised</td>
<td>55.89 ± 0.26</td>
<td>41.67 ± 0.33</td>
<td>75.94 ± 0.41</td>
</tr>
<tr>
<td>LP</td>
<td>46.20 ± 0.76</td>
<td>38.43 ± 1.88</td>
<td>70.29 ± 0.81</td>
</tr>
<tr>
<td>MT</td>
<td>45.36 ± 0.49</td>
<td>36.08 ± 0.51</td>
<td>72.51 ± 0.22</td>
</tr>
<tr>
<td>LP+MT</td>
<td>43.73 ± 0.20</td>
<td>35.92 ± 0.47</td>
<td>72.78 ± 0.15</td>
</tr>
<tr>
<td>SWA</td>
<td>-</td>
<td>34.10 ± 0.31</td>
<td>-</td>
</tr>
<tr>
<td>HrOT (k-means)</td>
<td>44.28 ± 0.47</td>
<td>36.82 ± 0.33</td>
<td>73.84 ± 0.44</td>
</tr>
<tr>
<td>HrOT w/o CL</td>
<td>42.06 ± 0.62</td>
<td>35.57 ± 0.64</td>
<td>72.94 ± 0.35</td>
</tr>
<tr>
<td>Soft HrOT</td>
<td>40.66 ± 0.71</td>
<td>32.88 ± 0.36</td>
<td>68.94 ± 0.51</td>
</tr>
<tr>
<td>HrOT</td>
<td>38.98 ± 0.91</td>
<td>30.86 ± 0.56</td>
<td>67.66 ± 0.75</td>
</tr>
</tbody>
</table>

Figure 5.3: indicate the number of accurate predicted labels by HrOT and CNN.

on the entropy minimization [51]. This set of methods forces the model to produce confident predictions (i.e., low entropy). Similarly, once we use one-hot targets, we essentially encourage the network to produce more confident predictions compared to using soft-pseudo labels.

### 5.4.3 Contribution of Hierarchical Optimal Transport to SSL

CNN trained on a limited amount of the labeled data simply miss-classifies the unlabeled data. Instead, we use the OT to cluster the unlabeled data and then map them with the labeled measures for pseudo-labeling. To compare these two criteria for pseudo-labeling, we report the number of accurate pseudo-labels obtained for the unlabeled training data using HrOT and the CNN by its own prediction (i.e., ”13 layer” network). We experimentally show how HrOT has a greater positive influence on the training of CNN classifier. Essentially, this comparison allows us to know whether or not the CNN classifier can benefit from our method for producing pseudo-labels during the training, because, otherwise, the CNN can simply use its own predicted labels on the unlabeled training data during the training. To
Figure 5.4: (a, b) OT cost trend, (c, d) mapping measures to barycenters and each other.

Figure 5.5: Validation error for different clustering resolution.

indicate the efficiency of HrOT, we change the number of labeled data in the training set and report the number of accurately predicted pseudo-labels by our CNN, and HrOT on the remaining unlabeled training data. Figure 6.4(a)-6.4(c) show that, for SVHN and CIFAR-10/100, the labels predicted by HrOT on the unlabeled training data are more accurate than the CNN, which means that the entire CNN can better benefit from HrOT than the case where it is trained solely by its own predicted labels. Moreover, we monitored the trend of transportation cost between the labeled and unlabeled measures obtained by Eq. 8 during the training. Experiments on SVHN and CIFAR-10 in Figure 5.4(a) and Figure 5.4(b) show that this cost reduces as the images fed into the CNN are represented by a better feature set during the training. In Figure 5.4(c) and Figure 5.4(d), we also visualized barycenters of the clusters, and mapping between measures of two classes in CIFAR-10 (i.e., bird and frog) using Sinkhorn algorithm. The Figures show that measures of different classes are separated properly after the training. Here, filled and unfilled squares represent unlabeled, and labeled data, respectively and the color of squares indicates their label.
5.4.4 Clustering Resolution

We study effect of the clustering resolution on the performance of our model. We use 1k labeled data for SVHN and CIFAR-10, and 4k labeled data for CIFAR-100 and MiniImageNet datasets. We change number of the centriods during the clustering, and report the error on the validation set. The results in Figure 5.5(a)-5.5(d) indicates that our model benefits from over-clustering but intense over-clustering decreases performance. The rationale can be supported by SSL models based on consistency regularization [25, 27]. Specifically, if we intensively increase the number of the clusters, we only consider the global structure (or geometry) of the data in the label space, and then we ignore their local structure when transporting labels. This is not appropriate for SSL, since in such a case, we ignore the local consistency of data. However, if we cluster data in the label space via Wasserstein metric and then map them through HrOT, we exploit both local and global structure of the data in the label space during the transporting labels. To further validate this claim, we conducted following ablation study: in each batch, we solve an OT between labeled data \( \{x_l\}_{u=1}^m \) and unlabeled data \( \{x'_u\}_{u=1}^m \) directly and use the OT to assign pseudo-labels to the unlabeled data. We refer to this baseline as Direct-OT (DOT), our results in Table 5.1 & 6.1 indicate that mapping data directly using OT significantly reduces the performance of the CNN compared to our original model, HrOT which indicate importance of the clustering and considering hierarchical structure in OT for generating pseudo-labels in our SSL model.

In further study, instead of using OT to cluster, we use the regular k-means in our method. We refer to this baseline as HrOT (k-means). The compared results between HrOT and HrOT(k-means) in Table 5.1 & 6.1 shows the power of Wasserstein-metric in the k-means objective for finding unlabeled measures. Moreover, we ablated the center loss in (Eq. 9) to see the effect of this term as a local consistency regularizer in our SSL model (i.e., HrOT w/o CL in in Tables 5.1 & 6.1). The compared results with HroT in Tables 5.1 & 6.1 show that this term can have relatively a positive influence on the performance of our model.
5.4.5 Varying Labeled Data

We evaluate how varying the amount of initial labeled data degrades the performance of HrOT in the very limited label regime. We gradually increase the number of labeled data during the training and report the performance of our SSL method on the testing set. Here, we run our SSL method over 5 times with different random splits of labeled and unlabeled sets for SVHN and CIFAR-10/100, and report the results in Figure 5.6(a)-5.6(c). The results show that the performance of HrOT tends to level off as the number of labels increases.

5.5 Conclusion

We proposed a method which leverages optimal transport to train a CNN classifier in an SSL manner. We used the Wasserstein barycenters of the unlabeled data to identify the measures in the unlabeled set. Then, we used hierarchical optimal transport to map measures from the unlabeled set to measures in the labeled set with a minimum amount of the total transportation cost in the label space. Based on this mapping, pseudo-labels for the unlabeled data were inferred, which were then used along with the labeled data for training the CNN. Finally, we experimentally evaluated our SSL method to indicate its potential for leveraging the unlabeled data when labels are limited during the training.
Chapter 6

Self-Supervised Learning and Optimal Transport for Semi-Supervised Learning

The goal is to use Wasserstein metric to provide pseudo labels for the unlabeled images to train a Convolutional Neural Networks (CNN) in a Semi-Supervised Learning (SSL) manner for the classification task. The basic premise in our method is that the discrepancy between two discrete empirical measures (e.g., clusters) which come from the same or similar distribution is expected to be less than the case where these measures come from completely two different distributions. In our proposed method, we first pre-train our CNN using a self-supervised learning method to make a cluster assumption on the unlabeled images. Next, inspired by the Wasserstein metric which considers the geometry of the metric space to provide a natural notion of similarity between discrete empirical measures, we leverage it to cluster the unlabeled images and then match the clusters to their similar class of labeled images to provide a pseudo label for the data within each cluster. We have evaluated and compared our method with state-of-the-art SSL methods on the standard datasets to demonstrate its effectiveness.
6.1 Introduction

CNN models have enabled breakthroughs in computer vision and machine learning. However, training a CNN model relies on a large-scale annotated datasets which are usually tedious and labor intensive to collect [17]. Considering the vast amounts of unlabeled data available on the web, the idea to use the unlabeled data without human effort to annotate them has become very appealing [12,16]. In this work, we tackle the challenge of deep SSL, the task of which is to use the unlabeled data in conjunction with the labeled data to train a better CNN classifier. Conventionally, we are given a dataset $D = D_l \cup D_u$ where the data in $D_l$ are annotated by labels while the data in $D_u$ are not. The goal is to train a CNN classifier on the known categories in $D_l$ using the data in $D$. The test data involves only the classes that are present in $D_l$. The main challenge in SSL is to efficiently leverage the unlabeled $D_u$ to help learning on $D_l$. To make use of unlabeled data in the general setting of SSL challenge, there are two fundamental assumptions that must be taken into the consideration [16]: 1) We assume that labeled and unlabeled data come from the same or similar underlying distribution and there is no class distribution mismatch between the labeled and unlabeled sets. 2) We presume that the underlying distribution of data has some structure. SSL algorithms considers at least one of these structural assumptions: consistency, manifold and cluster.

In consistency assumption [26, 157–160], data samples in a small neighbourhood have the same class label. In cluster assumption [28,161–163], data tends to construct discrete clusters in some geometric sense, and data within the same cluster are more probably to have the same class label. In manifold assumption [31,164,165], data lie in the neighbourhood of a low-dimensional and well-defined manifold which can be classified by meaningful distances on the manifold. For all of these assumptions, it is important to consider the geometry of the data when designing an SSL method. For example, popular mean teacher [44] and $\pi$-model [41] leverage different data augmentations approaches, each of which uses a different strategy to explore the local geometry of the labeled data for generating new data.

Recently, the theory of Optimal Transport (OT) [123,124] is used as a tool in machine learning algorithms to consider the geometry of the data. For example, the Wasserstein
distance in OT uses the geometry of the metric space to provide a meaningful distance between two distributions even if the supports of these distributions do not overlap. This property of the Wasserstein distance has made it useful and practical for many computer vision and machine learning applications such as clustering [135-138], generative models [52,160], loss function [128], semi-supervised learning [163,167-170], and domain adaptation [129-134].

In this work, we propose a new SSL method based on the Wasserstein metric which follows the general assumptions in SSL. Inspired by the effectiveness of Self-Supervised learning in many tasks including SSL [171-173], we first pre-train our CNN using a self-supervised learning method, MoCo v2 [174-176]. This process potentially enforces a clustered structure in the feature space for the unlabeled data which motivates us to perform a clustering on the feature of unlabeled data and then infer a pseudo-label for them.

Specifically, using the self-supervised pre-training on the CNN, we make a cluster assumption about the unlabeled data in which clusters are identified by the Wasserstein barycenter of the unlabeled data. Then, we leverage the Wasserstein metric to match the clusters of unlabeled data to their most similar classes of labeled data to provide pseudo-labels for the unlabeled data. Here, the Wasserstein distance is a measure of similarity between two sets of data points where one of them contains labeled data while the other one consists of unlabeled data. This matching is based on the assumption that the labeled and unlabeled data within the same class have the same or similar distribution. Therefore, we would expect that the similarity between two sets of data which come from the same or similar distribution is more than the case where these sets of data come from completely two different distributions. Finally, depending on the matching, we infer a pseudo label for the unlabeled data within each cluster, which are used along with the initially labeled data to train our CNN classifier [177].

6.2 Related Work

6.2.1 Semi Supervised Learning for Deep Models

There are many SSL algorithms in the literature [16,24,33]. However, we briefly review the methods based on the pseudo labeling and consistency regularization which have been
incorporated with deep learning models.

**Pseudo-Labeling** was initially proposed in [50]. In SSL models based on the pseudo-labeling, the model usually relies on its own prior belief about the label of unlabeled data to obtain supplementary information over the course of training [30,50,155,178,179]. The main drawback of these methods is susceptibility to confirmation bias such that the model is confident about its incorrect prediction, and then overfits to incorrect pseudo-labels during the training [180]. Therefore, in these models, the incorrect pseudo-labels not only can not provide useful information during the training but also error of the model’s prediction is accumulated in the model and results in overfitting. This downside even gets worse in cases where the discrepancy between the domain of the unlabeled data is significant from that of labeled data.

**Consistency-based SSL models** perform based on the assumption that the model should be generally consistent with its predictions between a given data and its meaningfully-distorted versions [30]. This simple criterion on the models output has provided promising results in the SSL literature such as stochastic perturbations models [43], π-model [41], mean teacher [44], and virtual adversarial training (VAT) [45], Mixmatch [158], Remixmatch [157], and Fixmatch [159]. The primary idea in stochastic perturbations and π-model was initially proposed in [46] and is known as pseudo-ensembles. The pseudo-ensemble regularization techniques usually perform in such a way that under realistic perturbations of input $x$ ($x \sim x'$), the prediction of the model $g(x, \theta)$ should not vary drastically. This objective is achieved by considering a weighted loss term such as $d(g(x, \theta), g(x', \theta))$ during the training of model, where $d(., .)$ denotes MSE or KL divergence which calculates a distance or divergence between outputs of the prediction function. The main problem in pseudo-ensemble approaches, including π-model is that they highly depend on a likely unstable prediction, which can instantly deviate significantly over the course of training.

To solve this issue, two approaches including temporal ensembling [41] and mean teacher [44], were introduced to achieve a more stable target output $g'(x, \theta)$. In temporal ensembling, the model uses an exponentially accumulated average of outputs, $g(x, \theta)$, to produce a smooth and consistent target output while in mean teacher, the model uses a prediction function parametrized by an accumulated average of the model parameters $\theta$ during the training.
Contrary to the stochastic perturbation methods mentioned earlier, VAT initially estimates a small perturbation \( r \) to add it to \( x \) which drastically changes the model prediction, \( g(x, \theta) \). Then, a consistency regularization term, \( d(g(x, \theta), g(x + r, \theta)) \) is considered as a loss term during the training.

Following the advance in consistency regularization, and pseudo-labeling for SSL, MixMatch integrates data augmentation, consistency regularization [43], entropy minimization [47], and mixup [181]. ReMixMatch enhanced on MixMatch by including augmentation anchors and distribution alignment. Augmentation anchors are performs similar to pseudo-labeling. FixMatch which is the state-of-the art and the most recent approach in this line of research combines consistency regularization, and pseudo-labeling with a threshold of confidence on the output of the model.

### 6.2.2 Self-Supervised Learning

The idea behind self-supervised learning (Self-SL) is to take large amount of readily and available unlabeled data and use it to understand itself [174–176, 182, 183]. Generally, the purpose of Self-SL for images is to create image representations that are semantically meaningful via pretext tasks that do not need human-annotations for a large training dataset. Pretext tasks usually guide the model towards learning meaningful representations that are covariant with image transformations such as rotations [184], and jigsaw transformations [182], and affine transformations [185, 186]. Recently, it has been shown that Self-SL approaches can be simply used to leverage all unlabeled data for learning and can be incorporated by SSL models [171]. For example, the work in [171] demonstrated that integrating simple Self-SL losses such as rotation is useful for a SSL approach.

### 6.3 Wasserstein Distance

For any subset \( \theta \subset \mathbb{R}^d \), assume that \( \mathcal{P}(\theta) \) represents the space of Borel probability measures on \( \theta \). The Wasserstein space of order \( k \in [1, \infty) \) of probability measures on \( \theta \) is defined as follows: \( \mathcal{P}_k(\theta) = \{ \mathbb{P} \in \mathcal{P}(\theta) : \int ||x||^k d\mathbb{P}(x) < \infty \} \), where \( ||.|| \) is the Euclidean distance in \( \mathbb{R}^d \). Let \( x \sim \mathbb{P} \in \mathcal{P}(\theta) \), \( y \sim \mathcal{Q} \in \mathcal{P}(\theta) \) and \( J(\mathbb{P}, \mathbb{Q}) \) denote all the joint distributions \( J \) for...
$(x, y)$ on $\theta \times \theta$ that have marginals $\mathbb{P}$ and $\mathbb{Q}$ for $x$ and $y$, respectively, and also assume that \(\delta(x, y)\) is a distance measure between two instances $x$ and $y$. Then, the Wasserstein distance is defined as follows:

$$W_k(\mathbb{P}, \mathbb{Q}) = \left( \inf_{J \in \mathcal{J}(\mathbb{P}, \mathbb{Q})} \int \delta(x, y)^k dJ(x, y) \right)^{1/k},$$

(6.1)

where $k \geq 1$. In case $k = 1$, this is also called the Earth Mover distance. The term $J(x, y)$ can be considered as a plan that transports a unit of mass from location $x$ to another location $y$ such that the marginal constraints are satisfied. The minimizer $J^*$ in Eq. (1) is called the optimal transport plan. In the case where transporting cost of a unit of mass from $x \sim \mathbb{P}$ to $y \sim \mathbb{Q}$ is equal to $\delta(x, y)^k$, then $W_k(\mathbb{P}, \mathbb{Q})$ is the minimum expected transportation cost. The Kantorovich-Rubinstein dual theorem [123] indicates that in the special case where $k = 1$, the Wasserstein distance has a closed form of an integral probability metric as follows:

$$W_1(\mathbb{P}, \mathbb{Q}) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}}[f(x)] - \mathbb{E}_{x \sim \mathbb{Q}}[f(x)],$$

(6.2)

where the supremum is over all 1-Lipschitz functions $f: \mathcal{X} \to \mathbb{R}$, and Lipschitz semi-norm is defined as follows: $\|f\|_L = \sup_{x, y} |f(x) - f(y)|/\delta(x, y)$.

### 6.4 Wasserstein Barycenter

Wasserstein Barycenter was initially introduced by [125], and provided an efficient role in clustering methods based on OT [135-138]. Let $\theta$ denote a Polish space, and $P(\theta)$ represent the space of probability measures on this space. Moreover, let’s assume that we are given $M \geq 1$ probability measures $\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_M \in P(\theta)$ with finite second moments, then the Wasserstein barycenter of these measures is defined as follows:

$$B(\tilde{\mathcal{P}}) = \inf_{\tilde{\mathcal{P}} \in P(\theta)} \frac{1}{M} \sum_{i=1}^{M} W_2^2(\tilde{\mathcal{P}}, \mathcal{P}_i),$$

(6.3)

it has been demonstrated by [148] that the problem of exploring Wasserstein barycenter on the space of $P(\theta)$ in Eq. (3) boils down to search only on a reduced space $\mathcal{O}_r(\theta)$ where
Figure 6.1: An illustration of mapping clusters to classes.

\[ r = \sum_{i=1}^{M} e_i - M + 1 \] and \( e_i \) is the number of elements in \( P_i \) for all \( 1 \leq i \leq M \). Moreover, several practical and effective algorithms have been recently proposed in \([125, 135, 149]\) that provide proper local solutions for the Wasserstein barycenter problem over the space of \( O_r(\theta) \). These algorithms such as the one in \([135]\) have been a building block for many interesting clustering algorithms based on OT such as \([137]\).

### 6.5 Proposed Method

Here, we describe the outline of our SSL method. Our SSL model contains three steps as follows: in step (1), we initially pre-train our CNN using a self-SL method on the unlabeled data and then fine-tune it using the initially labeled data. This operation potentially encourages the CNN model to construct cluster structure when representing the data. For example, in Self-SL based on contrastive learning paradigm \([174, 176]\), the goal is to learn similarities/dissimilarities such that the model is able to understand that the similar data should be closer to each other while dissimilar data should be far away from each other in terms of their representations. Therefore, pre-training the CNN motivates us to make a cluster assumption on the unlabeled data and then annotate each cluster with a unique pseudo-label.

In step (2), we use Wasserstein distance as a metric of similarity between two discrete probability measures to match each cluster of the unlabeled data to the most similar class.
of the labeled data for pseudo-labeling (see Fig. 6.1). This pseudo-labeling is based on the SSL assumption in which the labeled and unlabeled data within the same class should come from the same or similar distribution. Thus, we would expect that the similarity between two clouds of data which come from the same or similar distribution is more than the case where these clouds come from completely two different distributions.

Finally, in step (3), we use the unlabeled data annotated with the pseudo labels obtained from step (2) in conjunction with the initially labeled data to train our CNN classifier.

### 6.5.1 Self-Supervised Learning and Clustering via Wasserstein Barycenter

As discussed earlier, in step (1), we initially pre-train our CNN model using a Self-SL paradigm on the unlabeled data to make a cluster assumption for them. Here, we use MoCo v2 Self-SL [176] as it is a strong and efficient Self-SL method. Specifically, we use SimCLR [175] style data augmentation for the unlabeled images in the contrastive loss, and follow the implementation details in MoCo v2 where we use a two-layers MLP on the top of the last feature layer to map image features to a 128 dimensions, and then use a momentum updated model to calculate the key features in the memory bank.

After pre-training, we use the Wasserstein metric to perform a clustering on the unlabeled features extracted from the network. Following the previous clustering method based on OT [135–138], here we relate the clustering algorithm to the problem of exploring Wasserstein barycenter of the unlabeled data to find the clusters underlying them. The K-means objective is an optimization problem that has come up in the quantization problem [150].

Given $n$ unlabeled data $\{x_1, ..., x_n\} \in \mathbb{R}^d$, suppose that these data are grouped into $k$ clusters where $k \geq 1$. The K-means algorithm aims to find a set $C$ which contains $k$ elements $\{c_1, ..., c_k\}$ that minimizes the following objective:

$$F(C) = \inf_{C} \frac{1}{n} \sum_{i=1}^{n} D^2(x_i, C), \quad (6.4)$$

let $\mathcal{P}_n = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}$ be a probability measure where $\delta_{x_i}$ is the Dirac function on $x_i$. Then, problem (4) is equal to exploring a probability measure $Q$ with $k$ finite atoms that minimizes
the following objective:

$$B(Q) = \inf_{Q \in \mathcal{O}_k(\theta)} \sum_{i=1}^{n} W_2^2(Q, P_n),$$

(6.5)

this optimization problem can also be thought as a Wasserstein barycenter problem when $M = 1$ in Eq. (3). From this prospective, as introduced by [135], the algorithm for exploring the Wasserstein barycenter is an alternative for the well-known Loyd’s algorithm to obtain local minimum for the K-means. In this work, we use [135] to find the Wasserstein barycenter of the unlabeled data for clustering.

6.5.2 Matching Clusters to Classes via WGAN

After clustering the unlabeled data, in step (2), we follow the cluster assumption in SSL where data within the same cluster more likely should have the same class label. Moreover, in the general setting of SSL, data within the same class in both labeled and unlabeled sets have the same or similar distribution. Therefore, by considering the Wasserstein distance as a metric of similarity between two discrete probability measures, label of each cluster can be predicted based on the closest Wasserstein distance that the cluster has with a class of labeled data in the labeled set. This is because we would expect that the similarity between two sets of data coming from the same or similar distribution is more than the case where they come from completely two different distributions. Since we usually deal with large scale datasets, and CNN model is usually trained by stochastic gradient descent, we follow the standard training procedure, and use an approach based on gradient descent [52, 133, 187] to compute the Wasserstein distance.

Suppose that $P_i = \frac{1}{n_i} \sum_{j=1}^{n_i} \delta_{x_{ij}}$ denotes a labeled discrete measure which is constructed by labeled data $x_{ij}$ belonging to the $i$-th class; and $Q_i = \frac{1}{n'_i} \sum_{j=1}^{n'_i} \delta_{x'_{ij}}$ denotes an unlabeled discrete measure which is constructed by unlabeled data $x'_{ij}$ belonging to the $i$-th cluster. In step (2) of our algorithm, we aim to match each of $Q_1, ..., Q_k$ to one of the labeled measures $P_1, ..., P_c$, so that we can infer a label for each cluster. Therefore, we use the empirical Wasserstein distance as a measure of similarity between each pair $(Q_i, P_j)$ to match the pairs. For example, if the labeled measure $P_m$ is the closest measure to the unlabeled measure $Q_i$, we annotate the data within the $i$-th cluster with label $m$. 
In our SSL method, we use the CNN pre-trained via Self-SL to extract the feature for a given sample. Given an image $x \in \mathbb{R}^{m \times n}$, the CNN as a function $f_n : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^d$ with parameters $\theta_n$ maps sample $x$ to a d-dimensional representation. Inspired by the Wasserstein Generative Adversarial Network (WGAN) [52], we use a critic layer to compute the Wasserstein distance between each pair $(\mathcal{Q}_i, \mathcal{P}_j)$. Given a feature $z = f_n(x)$ obtained by the CNN, the critic layer in our model learns a function $f_c : \mathbb{R}^d \rightarrow \mathbb{R}$ with parameters $\theta_c$ that maps a feature to a real number. Therefore, the Wasserstein distance between two discrete measures $\mathcal{P}_i$ and $\mathcal{Q}_j$, where $z = f_n(x)$, $z' = f_n(x')$, $x \in \mathcal{P}_i$ and $x' \in \mathcal{Q}_j$ can be calculated by using Eq. (2) as follows:

$$W_1(\mathcal{P}_i, \mathcal{Q}_j) = \sup_{\|f_c\|_L \leq 1} \mathbb{E}_{\mathcal{P}_i}[f_c(z)] - \mathbb{E}_{\mathcal{Q}_j}[f_c(z')]$$

(6.6)

By considering the parameterized class of critic functions $f_c$ are all 1-Lipschitz, we can then calculate the empirical Wasserstein distance by maximizing the critic loss $L_w$ with respect to parameters $\theta_c$ as follows:

$$L_w(\mathcal{P}_i, \mathcal{Q}_j) = \frac{1}{|\mathcal{P}_i|} \sum_{x \in \mathcal{P}_i} f_c(f_n(x)) - \frac{1}{|\mathcal{Q}_j|} \sum_{x' \in \mathcal{Q}_j} f_c(f_n(x')).$$

(6.7)

Now, we need to force the Lipschitz constraint. In WGAN [52], it is suggested to clip the weights of critic layer in a compact interval $[-c, c]$ after each gradient update. However, weight clipping causes some issues including capacity underuse, and exploding problems or gradient vanishing [166]. Therefore, we use the technique used [166] to force a gradient penalty $L_{\text{grad}}$ for critic parameters $\theta_c$ as follows:

$$L_{\text{grad}}(\hat{z}) = (\|\nabla_{\hat{z}} f_c(\hat{z})\|_2 - 1)^2,$$

(6.8)

where the features $\hat{z}$ on which to penalize the gradients are the features of the labeled and unlabeled data, and also the random points along the line between labeled and unlabeled pairs. Therefore, we can approximate the Wasserstein distance by optimizing the following
Table 6.1: Comparing test error between SSWPL and different baselines and SSL methods.

<table>
<thead>
<tr>
<th>Datasets</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
<th>SVHN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>250</td>
<td>4000</td>
<td>250</td>
</tr>
<tr>
<td>Supervised</td>
<td>56.85 ± 1.34</td>
<td>19.74 ± 0.23</td>
<td>39.47 ± 0.56</td>
</tr>
<tr>
<td>π model [36]</td>
<td>54.26 ± 3.97</td>
<td>14.01 ± 0.38</td>
<td>57.25 ± 0.48</td>
</tr>
<tr>
<td>Pseudo-Labeling [50]</td>
<td>49.78 ± 0.43</td>
<td>16.09 ± 0.28</td>
<td>57.38 ± 0.46</td>
</tr>
<tr>
<td>UDA [16]</td>
<td>8.82 ± 1.08</td>
<td>4.88 ± 0.18</td>
<td>33.13 ± 0.22</td>
</tr>
<tr>
<td>MT [44]</td>
<td>32.32 ± 2.30</td>
<td>9.19 ± 0.19</td>
<td>53.91 ± 0.57</td>
</tr>
<tr>
<td>MixMatch [158]</td>
<td>11.05 ± 0.86</td>
<td>6.42 ± 0.10</td>
<td>39.94 ± 0.37</td>
</tr>
<tr>
<td>ReMixMatch [157]</td>
<td>5.44 ± 0.05</td>
<td>4.72 ± 0.13</td>
<td>27.43 ± 0.31</td>
</tr>
<tr>
<td>FixMatch [159]</td>
<td>5.07 ± 0.33</td>
<td>4.31 ± 0.15</td>
<td>28.64 ± 0.24</td>
</tr>
<tr>
<td>SSWPL (k-means)</td>
<td>9.62 ± 0.47</td>
<td>7.74 ± 0.73</td>
<td>30.19 ± 0.35</td>
</tr>
<tr>
<td>SSWPL</td>
<td>4.11 ± 0.15</td>
<td>3.18 ± 0.09</td>
<td>26.52 ± 0.45</td>
</tr>
</tbody>
</table>

objective:

\[ W_1(P_i, Q_j) = \max_{\theta_c} (L_w - \alpha L_{\text{grad}}), \]  (6.9)

where \( \alpha \) is a coefficient that balances between \( L_w \) and \( L_{\text{grad}} \).

### 6.5.3 Total Loss for Training the CNN

In step (3), we remove two-layers MLP from top of the last feature layer which we used for Self-SL, and then place a softmax layer for the classification task. In this step, we aim to use the unlabeled data annotated by the pseudo labels in conjunction with the supervision signals of the initially labeled data to train our CNN classifier. Therefore, we use the regular cross entropy loss to train the parameters of our CNN as follows: Let \( \mathcal{X}_l \) be all of the labeled training data annotated by true labels \( \mathcal{Y} \), and \( \mathcal{X}_u \) be the unlabeled training data annotated by pseudo labels \( \mathcal{Y}' \), then the total loss function \( L(.) \), for training the CNN in SSL fashion is:

\[ L(\theta_n, \mathcal{X}_l, \mathcal{X}_u, \mathcal{Y}, \mathcal{Y}') = L_c(\theta_n, \mathcal{X}_l, \mathcal{Y}) + \lambda L_c(\theta_n, \mathcal{X}_u, \mathcal{Y}'), \]  (6.10)

where \( L_c(.) \) denotes cross entropy loss function, and \( \lambda \) is a hyperparameter that balances between two losses obtained from the labeled and unlabeled data. Our algorithm to train a CNN in the SSL fashion is described in Algorithm 1:
Algorithm 3: Self-Supervised Wasserstein Pseudo-Labeling

**Input:** $\mathcal{X}_l$, $\mathcal{X}_u$, $\alpha$, $\lambda$, $\beta_1$, $\beta_2$, $b$, $k$, $m$

1: **initialize:** critic layer $\theta_c$ with $\mathcal{N}(0, 0.001)$.
2: pretrain $\theta_n$ using MoCo v2 Self-SL.
3: repeat
4: $Z_l = \{z_i\}_{i=1}^m$, $Z_u = \{z'_u\}_{u=1}^m$: where $z_i = f_n(x_i)$.
5: $\{Q_1, \ldots, Q_k\} \leftarrow$ cluster $Z_u$ to $k$ groups.
6: $\{P_1, \ldots, P_c\} \leftarrow$ cluster $Z_l$ to $c$ classes.
7: for each $Q_i$ and $P_j$ do
8:   for $i = 1, \ldots, s$ do
9:     choose a batch: $\{x_i\}_{i=1}^b \subset P_j$, $\{x'_i\}_{i=1}^b \subset Q_i$,
10:    $z'_i \leftarrow f_n(x'_i)$, $z_i \leftarrow f_n(x_i)$,
11:   $\tilde{z} \leftarrow \{z'_i, z_i, \tilde{z}\}$: take sample $\tilde{z}$ randomly on lines between $z'_i$ and $z_i$ pairs,
12:   $\theta_c \leftarrow \theta_c + \beta_1 \nabla_{\theta_c} [L_w(z', z) + \alpha L_{\text{grad}}(\tilde{z})]$
13: end for
14: $S(i, j) \leftarrow L_w(P_i, Q_j)$, by Eq. (7)
15: end for
16: $\{y'_u\}_{u=1}^m \leftarrow$ pseudo label data within each cluster $Q_j$ with the most similar class (i.e., argmin $S(\cdot, j)$),
17: repeat
18:   choose a batch: $\{x_i\}_{i=1}^b \subset X_u \cup X_l$,
19:   $\theta_n \leftarrow \theta_n - \beta_2 \nabla_{\theta_n} [L(\theta_n, x, x', y, y')]$, by Eq. (10)
20: until for an epoch
21: until $\theta_n$ converge

### 6.6 Experiments

We carry out empirical analysis to show the effectiveness and benefit of our SSL algorithm over other state-of-the-art methods [36, 44, 50, 157–160]. Here, we perform following studies:

1) We report results for supervised-baseline where the CNN is only trained by initially labeled data, this is because the goal of SSL is to greatly improve the supervised-baseline. 2) We change number of the labeled and unlabeled data and report the results as an efficient SSL method should still perform well even by using a small number of labeled data and extra amount of unlabeled data. 3) We replace our OT-base clustering method with the popular k-means and report the results to demonstrate the importance of the Wasserstein metric in our SSL algorithm. 4) We conduct an analysis on the clustering resolution (i.e., $k$ in Alg 1) to see its importance in our model.

Following the compared methods, we have been consistent in CNN network and used the
We performed our experiments on the widely used CIFAR-10/100 [17], SVHN [99], and ImageNet [154] datasets. We note that in all of our experiments, we consider the general SSL setting where the labeled and unlabeled data coming the same or similar distribution, and a given unlabeled data belongs to one of the classes in the labeled set and there is no class distribution mismatch. Furthermore, for each of aforementioned datasets, we split the training set into two different sets of labeled and unlabeled data. We make sure that all classes are balanced such that each class should have the same number of labeled data.

For training, we set hyperparameter \( \lambda \) to 0.7 in all of our experiments. We use the regular SGD optimizer with momentum 0.9, and weight decay \( 10^{-4} \). We set the learning rate \( \beta_2 \) in Alg 1 to \( 3 \times 10^{-3} \) in all of our experiment. The batch size in the experiments \( b \) in Alg 1) is set to 128. We note that our batch size for training the CNN \( b \) is different from the batch size that we map the unlabeled data to the labeled data \( m \) in Alg 1). The batch size for mapping the unlabeled data to the labeled data is the size of initially labeled data \( |\mathcal{X}_l| \). In other words, each time, we select \( |\mathcal{X}_l| \) unlabeled data to cluster them. Then, we use WGAN to map these clusters to the groups of data formed by \( \mathcal{X}_l \).

The parameters of the network \( \theta_n \) in Alg 1) are initialized by pre-training via MoCo v2 Self-SL [176]. Here, we follow the implementation details from MoCo-v2 but we use a memory bank of size 16384. We initialized the parameters of the critic layer \( \theta_c \) in Alg 1) by sampling randomly from \( \mathcal{N}(0, 0.001) \). The critic layer parameters usually requires around 10 epochs \( s \) in Alg 1) to converge in our experiments but we set it to 20 epochs for a sufficient
optimalization guarantee for the parameters of the critic layer. For training the critic layer, the
learning rate is also set to $\beta_1 = 3 \times 10^{-3}$. Note that during the training of the critic layer, we
penalize the gradients not only at CNN outputs for the unlabeled and labeled data points
but also at random points along the line between pairs of labeled and unlabeled data points.
The coefficient $\alpha$ is set to 10 as is suggested in [166].

In our experiments, we use the regular data augmentation and standard data normalization
techniques. Specifically, for SVHN, we converted and normalized pixel intensity values of the
images to floating point values in the range of $[-1, 1]$. For the data augmentation, we only
applied random translation by up to 2 pixels. For CIFAR-10/100, we used global contrast
normalization. The data augmentation on CIFAR-10/100 are random translation by up to 2
pixels, random horizontal flipping, and Gaussian input noise with standard deviation 0.15.

### 6.6.1 Comparison

The goal in SSL is essentially to obtain a better performance when we use the unlabeled
data compared to the case where we use the labeled data alone. Thus, we report the
error rate of our ‘WRN-28-2’ for cases where we only use a limited amount of labeled
data (i.e., Supervised in Table. 6.1), and the case where we leverage the unlabeled data
using our SSL method called Self-Supervised Wasserstein Pseudo Labeling (SSWPL) in
Table. 6.1 Furthermore, we report the performance of the other SSL methods including $\pi$
model [36], Pseudo-Labeling [50], UDA [160], MT [44], MixMatch [158], ReMixMatch [157],
and FixMatch [159] in Table. 6.1 For comparison, we chose 250, and 4000 labeled images for
CIFAR-10, 2500, and 10000 labeled images for CIFAR-100, 250, and 1000 labeled images for SVHN. Here, the remaining images of the training set are used as the unlabeled images to train the network. We ran our SSL methods over 5 times with different random splits of labeled and unlabeled sets for each dataset, and we reported the mean and standard deviation of the test error rate in Table 6.1. The results on CIFAR-10/100 and SVHN datasets in Table 6.1 demonstrate the potential of SSWPL for using the unlabeled data in comparison to other state-of-the-art SSL algorithms.

6.6.2 Self-SL Contribution on Clustering

As discussed in step (1), we pre-train our CNN model using Self-SL method and then form the clusters on the unlabeled data. Here, we evaluate the importance of the Self-SL on the clustering performance in our model. This is because one may assume that pre-training of the CNN on the initially labeled data can also enforce a clustered structure in the feature space for the unlabeled data, so it is important to know the benefit of using Self-SL on the clustering performance which plays an essential role in our model. Therefore, we conducted experiments to compare two different cases where in case (1), we fine-tune the network using initially labeled data without considering the Self-SL while in case (2) we consider the Self-SL for clustering. To compare these two cases and indicate the positive influence of the Self-SL on clustering, we changed the number of initially labeled data in the training set and reported the number of accurately predicted pseudo labels using our SSL method in case (1) and (2) on the remaining unlabeled training data. The significant gap between case (1) and case (2) which are respectively indicated by SSWPL and w/o Self-SL in Fig. 6.2 show that for CIFAR-10/100 and SVHN datasets, the labels predicted by our SSL method on the unlabeled training data are more accurate in case (1) than case (2), which means that the entire CNN network can benefit from Self-SL.

6.6.3 Analysis in Limited Label Regime

Here, we investigate that how changing the amount of initially labeled data increase the accuracy of our SSL algorithm in the very limited label scenario, and also at which point our
SSL algorithm can recover the performance of training when using all of the labeled data in the dataset. To conduct this evaluation, we moderately increase the number of labeled samples during the training and report the performance of our SSL algorithm on the testing set. In this study, we ran our SSL algorithm over 5 times with different random splits of labeled and unlabeled sets for each dataset, and reported the mean and standard deviation of the error rate in Fig. 6.3. The results indicate that the performance of our SSL method on CIFAR-10/100 and SVHN inclines to converge as the number of initially labeled data increases.

### 6.6.4 Varying Number of the Clusters

We evaluate the role of the clustering resolution on the error rate of SSWPL. In this study, we use 500, 1000, and 4000 labeled images from the training sets of SVHN, CIFAR-10, and CIFAR-100 datasets, respectively. We change the number of the clusters in our model, and evaluate error of the model on the validation set. The results on SVHN and CIFAR-10/100 datasets in Fig. 6.4 demonstrate that as we increase the number of the clusters in our model, the model can benefit from it but performance of the model inclines to degrade as we largely perform over-clustering. The reason can be interpreted by SSL models based on consistency regularization [25–27]. In other words, if we significantly perform over-clustering, we basically disregard the local geometry or structure of the data when mapping clusters to the label classes using the Wasserstein metric which is not useful in SSL as we neglect the local consistency.
Furthermore, in our other studies, instead of using the Wasserstein metric in the K-means objective for clustering the unlabeled data, we used the generic K-means in SSWPL and reported the test error rate in Table 6.1. We call this baseline as SSWPL (K-means). The compared results between SSWPL and SSWPL (K-means) on SVHN, and CIFAR-10/100 datasets in Table 6.1 demonstrate advantage of leveraging the Wasserstein-metric in the K-means objective for our SSL model.

6.6.5 Results on ImageNet

We also conducted an experiment on the large-scale ImageNet dataset to evaluate the performance of our model when using unlabeled unlabeled data in a very limited label regime. Following the prior work [159, 160], we also used a ResNet-50 architecture and RandAugment [190] data augmentation technique to conduct our experiments. Here, we set the number of the clusters in our method to the number of classes (i.e., 1000). We used 10% of the training set as our initially labeled data and the remaining as the unlabeled data. We ran our model 3 times and reported the mean and standard deviation of top-1 (top-5) error rate. The supervised-baseline top-1 (top-5) error rate using 10% of the training data is 45.64 ± 0.83% (24.67 ± 0.32) while for our SSL model (i.e., SSWPL), FixMatch [159], and UDA [160] are 26.46 ± 0.44% (9.14 ± 0.26%), 28.54 ± 0.52% (10.87 ± 0.28%), and 31.22% (11.2%), respectively. These results indicate the efficiency and potential of our SSL method compared to other effective SSL approaches for the large-scale datasets.

6.6.6 Limitation, discussion and Future Work

As mentioned earlier, in this study we consider the general setting of SSL in the literature [16, 24, 33] where there is no class distribution mismatch and the main assumption is that the labeled and unlabeled data coming from the same or similar distribution. Specifically, every given unlabeled data should belong to one of the classes which present in the labeled set. However, the work [24] in Sec. 4.4 showed that using unlabeled data from the mismatched classes essentially has a negative impact on the performance of the studied SSL approaches compared to the case where these approaches do not use any unlabeled data at all. Likewise,
our method would also hurt the performance when using the unlabeled data from the mismatched classes. This is because our method will provide a pseudo-label for the unlabeled data whether they belong to the mismatched classes or not. Thus in such a case, our model predicts high confident but incorrect labels for the unlabeled data within the mismatched classes and then use them for training which causes a confirmation bias problem [180]. However, pre-training the network using the Self-SL approach on the unlabeled data as we used in our method potentially can cluster the unlabeled data from mismatched classes as good as the unlabeled data which are not from mismatched classes. Therefore, in such a case, we can perform the clustering approach on the entire data and then disregards the clusters which contain the unlabeled data from mismatched classes during the training. There are many methods in the literature [191–194] that are proposed to detect out of distribution samples which we can potentially use them to detect out of distribution clusters. We will consider this study as our future work.

6.7 Conclusion

We proposed a new SSL algorithm that uses the Wasserstein distance and Self-SL technique to provide pseudo labels for the unlabeled data to train a CNN classifier in an SSL fashion. In this work, after pre-training the CNN model using a Self-SL method, we made a cluster assumption about the unlabeled data and then used their Wasserstein barycenter to explore the clusters underlying them. In the next step, we used the Wasserstein GAN to match each of the clusters to the most similar class of labeled data so we can provide a unique label for the data within each cluster. Finally, we used all the unlabeled data annotated by pseudo labels in conjunction with the initially labeled data to train our CNN model. In this study, we conducted empirical analysis to demonstrate the potential and efficiency of our SSL algorithm for leveraging the unlabeled data when labels are limited over the course of training.
References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


