Learning Tree-Structured Composition of Data Augmentation

Anonymous authors
Paper under double-blind review

Abstract

Data augmentation is widely used in settings where one needs to learn a neural network given very little labeled data. Typically, a composition of several transformations is applied sequentially to transform a given sample. Existing approaches for finding a composition either rely on domain expertise, or involve solving a complex optimization problem. The key challenge is that for finding a composition of length d, the search space is k^d, given a list of k transformation functions.

In this paper, we focus on designing efficient algorithms whose running time complexity is much faster. We propose a top-down recursive algorithm to search inside the space of tree-structured composition (of the k transformations), where each tree node corresponds to one transformation. The tree structure can be viewed as a generalization of existing augmentation methods, such as the one constructed by SimCLR (Chen et al., 2020b). Our algorithm runs in time $O(2^d k)$, which is much faster than the worst-case complexity of $O(k^d)$ (as soon as k grows away from 2). We extend the algorithm to tackle data distributions with heterogeneous subpopulations by finding one tree for each subpopulation and then learning a weighted combination of the trees.

We validate the proposed algorithms on several graph and image data sets, including a multi-label graph classification data set we collected. The dataset exhibits significant variations in the sizes of graphs and their average degrees, making it ideal for studying data augmentation. On the graph classification data set, our proposed algorithms can reduce computation by 43% over several recent augmentation search methods while improving performance by 4.3%. Besides, extensive experiments in contrastive learning also validate the benefit of our algorithm. The tree structures allow one to interpret the relative role of each augmentation, for example, identifying the important transformations on small vs. large graphs.

1 Introduction

Data augmentation is a technique to expand the training data set of a machine learning algorithm by transforming data samples with a pre-defined transformation function. Data augmentation is widely used in settings where a minimal amount of data has been annotated, such as self-supervised learning (Xie et al., 2020; Chen et al., 2020b). A common practice is to apply a sequence of transformations (Ratner et al., 2017; Cubuk et al., 2019; Lim et al., 2019; Zhang et al., 2020), for instance, first "Rotate" then "Equalize" the histogram of the images, as discovered by AutoAugment (Cubuk et al., 2019). The ideal sequence of transformations heavily depends on the downstream application and varies between domains (e.g., from natural to medical images). Therefore, it would be desirable to have a generic procedure that, given a list of input transformations and a target data set of interest, finds the most beneficial composition of transformations for that data set.

Although various techniques have been introduced in prior work to overcome the computational challenge of finding transformation compositions, they often involve solving a highly complex optimization problem. In the worst case, if we have k transformation functions, and the goal is to find the best sequence of length d, the search space would have k^d possible choices. Despite extensive research, existing optimization algorithms typically suffer from a worst-case complexity of $O(k^d)$. This paper aims to revisit this computational issue in a range of settings of strong practical interest by designing search algorithms whose running-time complexity can be much faster.

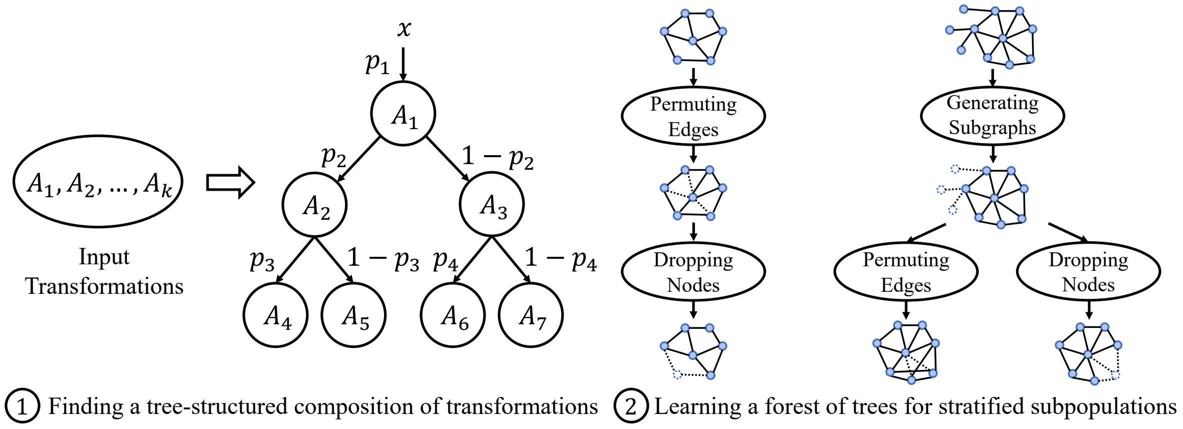


Figure 1: We illustrate the overall procedure of our algorithms. The input consists of k transformation functions, denoted as $A_1, \ldots, A_k$. Given a data set, one algorithm constructs a tree-structured composition of these transformations, as shown on the left. Given an input data x, A_1 is applied to map x to $A_1(x)$, with probability p_1; otherwise, no transformation is applied, and x remains unchanged. Let x' denote the output. In the next step, we will apply A_2 to x' with probability p_2, or A_3 to x' with probability $1 - p_2$, etc. The second algorithm will first partition the entire data set into a few groups, e.g., by the sizes of graphs, as illustrated above. Then, the first algorithm is applied to learn one tree for each partition. These trees are weighted jointly to form a "forest" as the final augmentation scheme. A byproduct is that we can now measure the importance of each transformation in the tree of each group. For example, we find that permuting edges by randomly adding or deleting a fraction of edges works best for small graphs. For large graphs, generating a subgraph by simulating a random walk works better. Notice that in an augmentation tree if a branch only has a single child node, it means if the transformation is not applied, then we will not change the input or use any augmentation (which is the same as applying $A(x) = x$).

As an example, we consider a data set of proteins with the goal of predicting their biological activity (function), a task relevant to biological discovery (Radivojac, 2022) and precision medicine (Rost et al., 2016). Each protein is given as a graph, with nodes corresponding to amino acid residues and edges indicating spatial proximity between the residues in a protein's 3D structure. Protein function prediction can be seen here as a graph-level multi-label classification problem (Clark & Radivojac, 2011), where the presence of a specific function indicates a binary label and proteins can have multiple functions. Since the graphs can vary considerably in size and degree distribution, finding the right augmentation for this type of data set is particularly challenging; see also Gao et al. (2023) for a detailed study on image data sets with stratified subpopulations.

Existing methods for finding composition either rely on domain expertise to identify the most relevant transformations first and then find the sequence or involve tackling a hyper-parameter optimization problem over the search space. For instance, reinforcement learning-based methods are one powerful family of methods (Cubuk et al., 2019; Luo et al., 2023), which define a reward function given an augmentation composition, such as the validation accuracy, and then search over the discrete space of compositions to optimize the reward function. Note that these methods require exploring a search space of size k^d (for finding a sequence of length d). Therefore, their worst-case running time complexity can still be $O(k^d)$.

The contribution of this paper is designing a faster algorithm to find a binary tree composition of k transformations of depth d, whose running time complexity is instead only $O(2^d k)$ provably. Surprisingly, empirical findings show that this complexity reduction does not come at the cost of downstream performance. The algorithm conducts a top-down, recursive search to construct a binary tree, where each node of the tree corresponds to one transformation. This tree structure leads to a natural notion of importance score for each transformation, analogous to the feature importance score used in tree-based methods. Additionally, the algorithm can be used to tackle the abovementioned scenario where the underlying data involves a mixture of stratified subpopulations. See Figure 1, which illustrates our algorithms.

We conduct extensive experiments across several data sets to validate our proposed algorithms. First, we apply our algorithm to a newly collected graph classification data set generated using AlphaFold2 protein structure prediction APIs (Jumper et al., 2021), containing 20,504 human proteins and 1,198 types of protein functions listed in Gene Ontology (Ashburner et al., 2000). Compared to existing data sets in Borgwardt et al. (2005) and Hu et al. (2020a), our data set is based on a newer source and contains a broader set of protein functions. We find that our algorithm can outperform RandAugment which does not search for composition by **4.3**%. Compared to recent augmentation optimization methods such as GraphAug (Luo et al., 2023), our algorithm reduces its runtime by **43**% and achieves **1.9**% better performance. Second, we evaluate our algorithm in contrastive learning settings. Compared to the augmentation method of SimCLR Chen et al. (2020b), the performance is on par with natural images such as CIFAR-10. For medical images, our algorithm can find a better augmentation scheme that outperforms SimCLR by **5.9**%. We have also tested our method on graph contrastive learning. Ablation studies justify the design of our algorithm. For details, see the experiment section.

To summarize, we list the main results of our paper as follows:

- We design an algorithm whose worst-case running time complexity is $O(2^d k)$, where k is the number of input transformations and d is the length of the composition. This is a significant improvement compared to the worst-case complexity of $O(k^d)$ (e.g., when $k > 10$ and $d > 3$).
- We extend the algorithm to tackle data distributions that involve a mixture of heterogeneous subpopulations, by first partitioning the data into several groups (e.g., according to the graph sizes). The algorithm is tested on a newly collected graph classification dataset and a few other benchmark data sets, reducing runtime while showing improved performance compared to several recent augmentation search methods.
- We construct a new multi-label graph classification data set, which contains a wide spectrum of graph sizes and degrees, making it ideal for studying data augmentation.

1.1 Related Work

We discuss several optimization frameworks which are related to our work. Besides the methods discussed so far, optimization frameworks such as minimizing the distance between augmented and unaugmented images (Hataya et al., 2020), gradient alignment (Zheng et al., 2021), and bilevel optimization (Lin et al., 2019; Hataya et al., 2022) have been used. Yang et al. (2022) propose an adversarial training objective to find hard positive examples. Benton et al. (2020) and Hounie et al. (2023) aim to optimize the invariance of a model's predictions for randomly sampled augmented inputs. The tree structures proposed in our work can also be used in these optimization frameworks.

Besides supervised learning, data augmentation is an essential component in contrastive learning (You et al., 2020). The invariance added by data transformation methods can be used as a regularizer in self-supervised learning methods (Xie et al., 2020). It is interesting to see if the tree composition can be used as a stronger form of regularization in self-supervised learning.

There are also theoretical analyses to explain the benefit of data augmentation methods. Wu et al. (2020) categorize the generalization effects of linear transformations using a bias-variance decomposition and study their effects on the ridge estimator in an over-parametrized linear regression setting. Yang et al. (2023) analyze different usage of augmented data and show that using the same augmented data, data augmentation consistency regularization can yield smaller generalization errors than empirical risk minimization on augmented data in linear regression settings.

There are studies on shifts in graph sizes, by accounting for local structures (Yehudai et al., 2021), or adding regularization on the representation distance between different sizes (Buffelli et al., 2022). Our work contributes to this literature by designing a generic, efficient data augmentation search method. There are also studies on using hyper-order structures in designing graph neural networks Chien et al. (2022). It is conceivable that the proteins graphs we constructed may involve such interactions and this would be another interesting direction for future research.

It would be interesting to examine theoretical justifications for the tree construction in the context of data augmentation. This likely requires new techniques, since theoretical results in the data augmentation literature are scarce, even though there has been much progress in developing guarantees for training neural networks in the past few years. Part of the challenge is that the data augmentation training paradigm violates the independent sampling assumption typically required in the theoretical literature. For instance, suppose one would like to understand how applying a sequence of transformations, like rotation, cropping, etc, to an image, affects the downstream performance. This would require modeling such transformations within the data augmentation. There have been few developments in this direction, for instance, work by Chen et al. (2020a) develops a group-theoretical framework modeling data augmentation. However, that work's result only applies to a single transformation, and it does not work for the composition of multiple transformations. Another recent work by Shao et al. (2022) develops a PAC-learning framework for transformation-variant data sets. Their algorithm relies on a counting argument and is thus not very practical.

We note that data augmentation has generally been quite helpful in practice. As hinted above, various transformations add invariance to the data set, which can, in turn, improve generalization, and this can be fleshed out in a high-dimensional linear regression setting Wu et al. (2020). There are studies (albeit in a very different setting) on the approximation ratio of greedy search in constructing trees, e.g., Adler & Heeringa (2008) and Gupta et al. (2017). It is an interesting research direction to examine these results in the context of data augmentation.

Organization. We start by giving some context for our problem setup in Section 2. Then, we will describe the proposed algorithms in Section 3. We will also give some examples of the algorithms to illustrate their design. Next, we present the empirical evaluations in Section 4, with a detailed comparison to existing methods. We provide supporting empirical and theoretical analysis in the Appendix.

2 Learning Composition of Data Augmentation

This section describes the problem setup for learning the composition of transformations. Consider a prediction problem where the goal is to map an input feature vector $x \in \mathcal{X}$ to a label $y \in \mathcal{Y}$. We have access to a data set $\hat{P}$, which includes a list of examples (x, y) drawn independently from an unknown distribution $\mathcal{P}$ over $\mathcal{X} \times \mathcal{Y}$. Given k transformation functions, denoted as $A_1, \dots, A_k$, each transforms an input feature vector with a predefined transformation function $A_i : \mathcal{X} \to \mathcal{X}$. The problem is to find a composition Q of (a subset of) the k functions, such that a model f_θ will be learned to minimize the loss of the augmented examples with a loss function ℓ. For example, in the case that ℓ is the cross-entropy loss over C classes, ℓ is a function that maps from $\mathbb{R}^C \times \mathcal{Y}$ to $\mathbb{R}$. We will consider Q as a probabilistic distribution over sequences of compositions. Denote τ as a sequence of transformations sampled from the distribution Q. The learning objective can be written as follows:

$$L(f_\theta; Q) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{P}} \mathop{\mathbb{E}}_{\tau\sim Q} [\ell(f_\theta(\tau(x)), y)].$$

When the underlying distribution involves several groups of stratified subpopulations, we can write the above objective as a mixture of losses over the groups. Let m denote the number of groups and let $\mathcal{G} = \{1, \dots, m\}$ denote the set of group labels. Let P_g denote the data distribution of $\mathcal{P}$ restricted to group g. For our context, it is sufficient to think that the group labels are available during training and testing time. Thus, the learning objective becomes

$$L(f_\theta; Q) = \sum_{g=1}^{m} q_g \cdot L_g(f_\theta; Q), \text{ where } L_g(f_\theta; Q) = \mathop{\mathbb{E}}_{(x,y)\sim P_g} \mathop{\mathbb{E}}_{\tau\sim Q} [\ell(f_\theta(\tau(x), y)], \tag{1}$$

and q_g denotes the proportion of examples coming from group g. Besides this empirical risk minimization setup, the min-max optimization has been studied before (Sagawa et al., 2020). The same ideas described later can be extended to this min-max formulation.

Compositionality. One form of compositionality is a probabilistic sequential composition. We illustrate an example of sequential composition used in SimCLR (Chen et al., 2020b) in Figure 2. This composition first applies random cropping, then color distortion, followed by Gaussian blurring. It is also a common practice to assign a probability value to apply each transformation in the literature (see, e.g., Cubuk et al. (2019)). Specifically, consider a sequence of length d, denoted as $(A_1, p_1), (A_2, p_2), \dots, (A_d, p_d)$, where each A_i is associated with a probability $p_i \in [0, 1]$. The transformations are applied sequentially from the first to the last, each with probability p_i. For example, if A_1 is not applied, then the input x remains as input to the next node, similar to a skip connection. And so on.

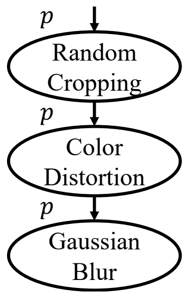


Figure 2: Illustrating a sequential augmentation scheme from SimCLR (Chen et al., 2020b).

In this paper, we will consider the family of tree-structured compositions, which naturally generalizes the sequences. We define the composition in the form of a binary tree. For a tree of depth d, there are $2^d - 1$ nodes. Each node is associated with an augmentation A_i and a probability value p_i, indexed by $i = 1, \dots, 2^d - 1$. Given such a tree, we apply its augmentations from the root to a leaf node. After applying an augmentation A_i, the augmentation proceeds to either the left or right node. Thus, the sum of probabilities for applying its left and right nodes equals one.

Computational costs. We now elaborate on the computational cost of existing methods. AutoAugment (Cubuk et al., 2019) uses reinforcement learning as the search method to find a composition that optimizes the validation performance. This method takes many GPU hours to search for a composition with depth $d = 2$ out of $k = 16$ augmentations on CIFAR-10. Fast AutoAugment (Lim et al., 2019) uses Bayesian optimization as the search method. These methods have the worst-case running time complexity of $O(k^d)$. Instead of searching in a discrete space, Chatzipantazis et al. (2023) parameterizes the space of compositions with $O(k^d)$ continuous variables, then uses stochastic gradient descent to optimize the model and the variables jointly.

3 Our Proposed Algorithms

We now present our algorithm for learning tree compositions. We design an algorithm that conducts a greedy search from the root to the leaves. We provide examples to validate this algorithm on benchmark data sets. Next, we extend the algorithm to a setting involving several subpopulations. We will find a weighted combination of the tree structures from each group to capture cross-group transferability.

3.1 Learning Tree-Structured Composition

We consider a top-down binary search procedure inspired by the *recursive binary splitting* of building a decision tree to reduce the complexity of constructing a tree-structured composition. The procedure searches from the root of a tree and iteratively finds one augmentation in one of the empty leaf nodes. A node would indicate a new branch of two nodes to build the tree further.

For a particular node i in the tree, we search for the choice of A_i and p_i that leads to the best improvement in the validation performance, such as cross-entropy loss. Each search iterates over all possible transformations $A_1, \dots, A_k$ and a discrete list of probabilities. To clarify, we include $A(x) = x$, namely identity mapping, as one of the k transformations. Naively optimizing the validation performance requires training $O(k)$ models for each choice and choosing one via cross-validation. This can also be expensive when k is large.

We make the search in each step more efficient by using a density matching technique to avoid repeatedly training multiple models. This is a frequently used hyper-parameter searching technique, such as in Fast AutoAugment (Lim et al., 2019). Specifically, we train one model with a current tree-structured composition, denoted as $f_{\theta^\star}$. Then, for every choice of A_i and p_i, we add them to the position i of the current composition and evaluate the performance of $f_{\theta^\star}$ on n augmented examples generated by applying the new augmentation composition to the validation set. Denote a tree-structured composition as T. The validation performance is:

$$L(T) = \frac{1}{n}\sum_{i=1}^{n} \mathop{\mathbb{E}}_{\tau\sim T}\left[\ell(f_{\theta^\star}(\tau(x)), y)\right]. \tag{2}$$

Thus, each search trains only one model and evaluates $O(k)$ times by adding each augmentation to position i in the current tree. We empirically find that the density matching technique finds the same tree-based composition of data augmentations in a protein graph and a wildlife image classification data set. In addition, we evaluate the relative residual sum of square error between the performance in Equation (2) with the true validation performance of training a model for each transformation choice. We find that the performance in Equation (2) only yields a relative error of 0.7% to the true validation performance.

We repeat the above step for the remaining leaf nodes in the tree:

- When a child node is added with transformation A_i and probability p_i, the other child node of the same parent node should find a transformation from one of the k input ones, and apply the chosen transformation with probability $1 - p_i$. In other words, the probability value $1 - p_i$ will be fixed.
- The process continues until the tree reaches a specified depth or the validation performance no longer improves by adding more augmentations on any remaining leaf nodes.

We summarize the procedure in Algorithm 1.

Algorithm 1 A top-down recursive search procedure

```
Input: $k$ transformation functions $A_1, A_2, \dots, A_k$, the training and validation sets
Require: Composition length $d$; A list of $H$ probability values
Output: A probabilistic tree-structured composition $T$ of $\{A_i\}_{i=1}^k$
1: Initialize $T = \{\}$, $V = \{1\}$, $L^{\text{val}}(T) = \infty$
2: while depth($T$) $< d$ and $V$ is not empty do
3:   Randomly choose $i \in V$, let $A_i,\ p_i,\ L_i^{\text{val}} \leftarrow$ Build-one-node($i,\ T$), then remove $i$ from $V$
4:   Add $A_i,\ p_i$ to position $i$ of $T$.
5:   if $L_i^{\text{val}} < L^{\text{val}}(T)$ then
6:     Update $L^{\text{val}}(T) = L_i^{\text{val}}$ and $V \leftarrow V \cup \{2i, 2i+1\}$
7:   end if
8: end while
9: return $T$
10: procedure Build-one-node($i,\ T$)
11:   Train a model with $T$ denoted as $\theta^\star$
12:   if the sibling node of $i$ is in $T$ then
13:     Let $p = 1 - p_k$ where $k$ is the index of the sibling node of $i$
14:     for $j \in \{1, \dots, k\}$ do
15:       Add $(A_j, p)$ to at position $i$ of $T$ and evaluate $L^{\text{val}}$ in equation (2)
16:     end for
17:   else
18:     for $j \in \{1, \dots, k\}$ and $p \in H$ do
19:       Add $(A_j, p)$ to at position $i$ of $T$ and evaluate $L^{\text{val}}$ in equation (2)
20:     end for
21:   end if
22:   return the $(A_j, p)$ with the lowest $L^{\text{val}}$
23: end procedure
```

Augmentation importance scores. The tree structures allow us to measure its interpretability, as in classical tree-based methods. We can measure which transformation is most helpful by how much it improves validation performance, analogous to the feature importance score in decision trees. Concretely, we can calculate the total decreased loss over the corresponding nodes of each transformation, since one augmentation can appear several times in the tree. A higher score indicates that a transformation contributes to reducing the loss more.

3.1.1 Running Time Analysis

We examine the running time of building one tree-structured composition. For a tree with depth d, the number of search steps is at most $2^d - 1$, which is the maximum size of this tree. Each search step involves training one model and iterating over all possible k augmentations and a list of probabilities. The overall procedure takes $O(2^d k)$ time to find a tree-structured composition.

Our algorithm reduces the complexity of learning a tree-structured composition with the top-down binary search procedure, which scales linearly to the number of augmentations k. In contrast, the complexity of previous methods scales with $O(k^d)$ since they consider the search space of all possible sequential compositions of augmentations. The comparison of the running time complexity with previous search methods is summarized in Table 1. In Section 3.1.2, we materialize the comparison in terms of GPU hours of the search methods.

Table 1: Summary of comparisons between our approach and previous optimization algorithms to search for compositions of data augmentations. We use d to denote a composition's length and k to denote the number of transformations to the input.

	Method	Running Time	Compositionality
AutoAugment (Cubuk et al., 2019)	Reinforcement Learning	$O(k^d)$	Sequential
Fast AutoAugment (Lim et al., 2019)	Bayesian Optimization	$O(k^d)$	Sequential
SCALE (Chatzipantazis et al., 2023)	Stochastic Gradient Descent	$O(k^d)$	Sequential
Algorithm 1 (Ours)	Binary Search	$\boldsymbol{O(2^d k)}$	Forest of Trees

3.1.2 Examples

Next, we give an example of running our algorithm in the setting of Chen et al. (2020b), before getting further into its details. We validate that our algorithm uses less runtime than previous search methods (without sacrificing performance). We give an example of our algorithm for training WideResNet-28-10 on CIFAR-10 and CIFAR-100, a setting considered in AutoAugment. We use the 16 types of transformations, each with five values of perturbation magnitude, as in AutoAugment (Cubuk et al., 2019), including ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, and Sample Pairing. We use eleven values of probabilities uniformly spaced between 0 and 1.0. Following the setting in AutoAugment, we use a reduced data set of randomly sampled 4,000 examples to search for the composition. After the search, we train a model on the full data set to report the test performance. For each algorithm, we report the runtime of the search procedure, evaluated on a single RTX 6000 GPU.

We report the results in Table 2. Our algorithm uses 4.7 GPU hours to search for the tree-structured composition, which reduces the runtime by 51% compared to the baseline methods. Meanwhile, the test performance of a model trained with the found composition remains within a comparable range to the baseline methods.

Next, we show an example of the augmentation composition for training ResNet-50 on CIFAR-10 with the infoNCE loss following SimCLR (Chen et al., 2020b). We use 7 augmentation methods considered in SimCLR, including Random Cropping, Cutout, Rotation, Color Distortion, Sobel filtering, Gaussian noise, Gaussian blur, five perturbation magnitudes, and eleven probabilities to apply the augmentations.

Figure 3 shows the tree-structured composition found by our algorithm. Interestingly, this tree shares three nodes, as in SimCLR augmentation, constructed by domain experts. The tree begins by applying random cropping and color distortion, similar to SimCLR augmentation. Then, the tree uses Gaussian blur as one branch but adds rotation as another branch. The performance of using these two is comparable when evaluating the trained model's contrastive features with logistic regression.

Table 2: We compare our algorithm with existing optimization algorithms for learning augmentation composition in training randomly initialized WideResNet-28-10 on CIFAR data sets. We report the runtime of the search process for the composition in terms of GPU hours using a single GPU. We also report the error rates on the test set, averaged over five random seeds.

	CIFAR-10		CIFAR-100	
	GPU hours	Error rates (%)	GPU hours	Error rates (%)
AutoAugment	5000	2.6±0.1	5000	17.1±0.3
Fast AutoAugment	19.2	2.7±0.1	19.6	17.3±0.2
SCALE	9.6	3.3±0.1	9.6	17.3±0.1
Algorithm 1 (Ours)	**4.7**	3.1±0.1	**4.7**	17.3±0.1

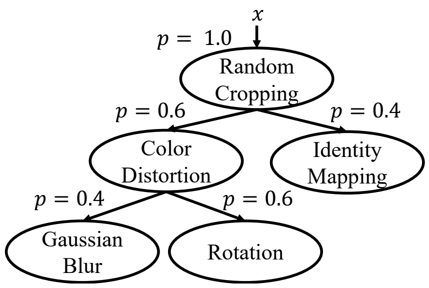


Figure 3: Illustrating the tree-structured composition returned by Algorithm 1 on CIFAR-10. On the right branch, no further transformation is applied after the identity mapping.

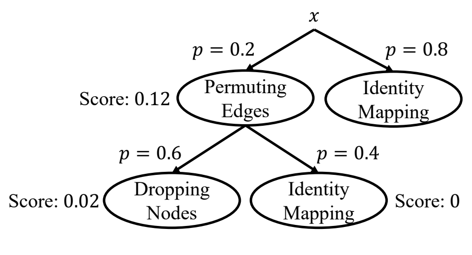
Lastly, we show that the top-down binary search procedure performs comparably to the exhaustive search (which enumerates every possible tree) of depth-2 trees (with three nodes). We notice that on several image and graph classification data sets, the test performance of our algorithm is only 0.4% below the exhaustive search (on average).

3.2 Learning a Forest of Trees

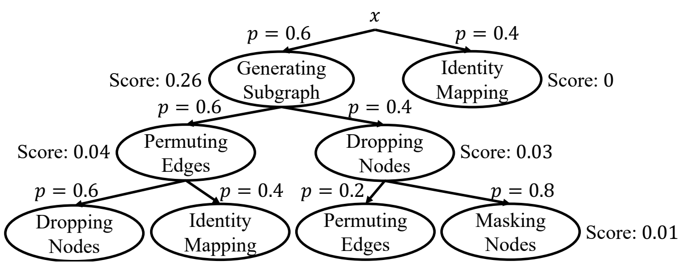
Our efficient search algorithm unlocks the application of learning data augmentation compositions for group shifts. We start by presenting a motivating example, showing that learning a single composition does not suffice for the case of group shifts. We consider graph classification as a case study involving heterogeneity in the graph sizes and degrees. We consider previously used graph transformations, including random modifications of graph structures, such as randomly dropping nodes, perturbing edges, or taking a random subgraph (You et al., 2020; Luo et al., 2023).

We test this hypothesis on a protein graph classification data set. The task involves predicting multiple binary-labeled classification problems, which aims to predict proteins represented as graphs to their protein functions, each treated as a zero-one label. Since labeling the protein functions requires domain expertise, we could only obtain about $4e^{-3}$ of all the labels. To tackle this issue, we will design data augmentation schemes to generate synthetic labels. Additionally, this data set exhibits large variations in graph sizes and degrees. The sizes of graphs range from 16 to 34,350, and the average degrees range from 1.8 to 9.4. In the experiment, we will divide the data set into 16 groups, which corresponds to 16 intervals of sizes and average degrees. We will use a graph neural network as the base model (Hu et al., 2020b). We consider four augmentation methods, including DropNodes, PermuteEdges, Subgraph, and MaskingNodes, as in (You et al., 2020). We consider eleven probability values for applying each augmentation uniformly spaced between 0 and 1.0.

We first find a tree-structured composition by minimizing the average loss. We apply this to each group and compare it with a model trained without augmentation. Applying this can even hurt performance for graphs with less than 200 nodes or an average degree of less than 0.1.



(a) Augmentation tree found on a small graph with the number of nodes less than 200.



(b) Augmentation scheme found on a large graph with the number of nodes larger than 600.

Figure 4: We illustrate that the augmentation found in each group varies between groups. On a protein graph classification data set, the augmentation tree on small graphs (left) involves fewer augmentation steps than those for large graphs (right). We also report each augmentation's (importance) score, computed from the validation set. We note that no further transformation is applied after the identity mapping, i.e., $A(x) = x$.

Next, we find one tree for each group, which differs across groups. In particular, for each group, we run Algorithm 1 to construct a specific tree. Also notice that the data we use for executing this step is separate across groups. Hence, there is no synchronization performed on different groups. In other words, the models we trained in each group will not be reused in training other groups (or in the weighted training step). This leads to some heterogeneity in the tree structure we obtain for different groups. To give an example, in Figure 4, we plot two compositions, corresponding to a group of the smallest sizes and another group of the largest sizes. The latter uses larger scales and one more step. Permuting Edges and Dropping Nodes are used on small graphs, which remove randomly sampled edges and nodes, respectively. All augmentations are used on large graphs, including Generating Subgraph, which extracts a subgraph generated by random walks, and Masking Nodes, which randomly masks a fraction of node attributes to zero. Moreover, we compute the importance of each selected augmentation. On small graphs, permitting edges has the largest importance score. In contrast, the transformation that takes a subgraph has the highest importance score on large graphs.

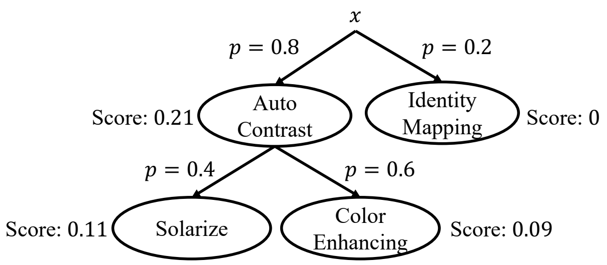
Furthermore, we provide another example on an image classification data set. This data set contains three groups of images with different colors and backgrounds. We illustrate the augmentation found for one group of colored images and another group of black-white images. We also notice a difference between the two augmentations. Transformations that change the color distributions are applied to the colored images, including Auto Contrast, Solarize, and Color Enhancing. On the other hand, transformations applied to black-white images change the grayscale or shape of the images, including Equalize, TranslateY, and Posterize. Moreover, on the colored images, the Auto Contrast transformation has the largest importance score. On the black-white images, the Equalize has the largest importance score. The observations validate our motivation that data augmentations vary across groups.

Learning the weight of each tree. With a tree-structured composition found for each group, next, we unify the compositions in a weighted training scheme. Specifically, we train a (new) single model f_θ with a weighted combination of per-group losses to optimize the average loss over the entire population. This problem can be cast into a bilevel optimization formulation:

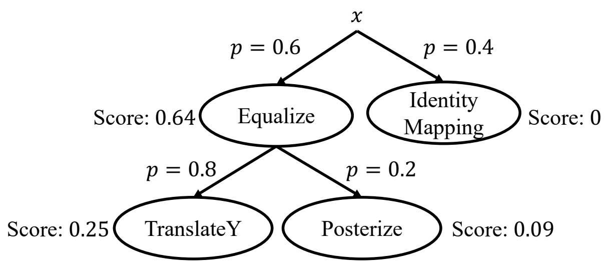
$$\min_{w,\theta} \hat{L}(f_\theta), \quad \text{such that} \quad \theta \in \arg\min_{\theta} \sum_{g=1}^{m} w_g \hat{L}_g(f_\theta; Q_g). \tag{3}$$

where the training loss of group i is associated with a weight w_i, and $\sum_{i=1}^{m} w_i = 1$. Given w, let

$$\theta^w \leftarrow \arg\min_{\theta} \sum_{g=1}^{m} w_g \cdot \hat{L}_g(f_\theta; Q_g).$$



(a) Augmentation found on colored images.



(b) Augmentation found on black-white images.

Figure 5: On an image classification data set, the augmentation found in each group varies between groups. The augmentation tree on colored images (left) involves different augmentations from the one for black-white images (right). We also report each augmentation's (importance) score, computed from the validation set. We note that no further transformation is applied after the identity mapping, i.e., $A(x) = x$.

Thus, we can rewrite objective (3) as a function of θ^w:

$$\min_{w \in \mathbb{R}^m} \hat{L}(f_{\theta^w}) = \sum_{g=1}^m q_g \cdot \hat{L}_g(f_{\theta^w}). \tag{4}$$

We then derive the gradient of the objective with respect to weights w. This is achieved by applying the chain rule and computing the gradient of θ^w through the implicit function theorem. For any $i = 1, \dots, m$, let the gradient to the weight w_i, denoted as d_i, be equal to

$$d_i := \frac{\partial \hat{L}(f_{\theta^w})}{\partial w_i} = -\left(\sum_{g=1}^m q_g \nabla_\theta \hat{L}_g(f_{\theta^w})\right)^\top H^{-1} \nabla_\theta \hat{L}_i(f_{\theta^w}; Q_i), \text{ where } H = \sum_{g=1}^m w_g \nabla_\theta^2 \hat{L}_g(f_{\theta^w}; Q_g). \tag{5}$$

The complete derivation can be found in Appendix A. Equation (5) can be viewed as a gradient similarity measure, normalized by the inverse of Hessian of the average loss.

Given the gradients of w and θ, we use alternating updates to minimize $\hat{L}$. Let the model parameters and weights at iteration t be $\theta^{(t)}$ and $w^{(t)}$. We apply the following two steps iteratively:

- With $w^{(t)}$, update the model parameters to $\theta^{(t+1)}$ by running α SGD steps on

$$\sum_{g=1}^m w_g^{(t)} \cdot \hat{L}_g(f_\theta).$$

- With $\theta^{(t+1)}$, obtain the derivative $d_i^{(t)}$ of $w_i^{(t)}$ from equation (5). Then, update $w_i^{(t)}$ via exponentiated gradient descent which operates in the space of a probability simplex, i.e., $w_i > 0$ for all i, and $\sum_{i=0}^m w_i = 1$:

$$w_i^{(t+1)} = \frac{w_i^{(t)} \exp(-\eta d_i^{(t)})}{\sum_{j=1}^m w_j^{(t)} \exp(-\eta d_j^{(t)})}. \tag{6}$$

Taken together, we summarize the complete procedure for learning a unified augmentation scheme for multiple groups in Algorithm 2. To recap, our algorithm learns a single augmentation scheme, which involves a probabilistic mixture of m trees (one learned from each group), as the final output of the algorithm.

We provide a generalization bound to justify the algorithm. At a high level, we show that the generalization error of the model trained by Algorithm 2 is bounded by two terms. The first term corresponds to the bias increase to transfer from a different data distribution, and the second term corresponds to the variance, which captures the benefit of knowledge transfer to the target group. The proof technique is based on carefully examining the bilevel optimization algorithm using covering numbers. We describe the complete statement of the generalization bound in Appendix C. The proof is based on the transfer exponent framework of Chen et al. (2022) and also the work of Hanneke & Kpotufe (2019).

Algorithm 2 **Learning a forest of trees for a mixture of stratified subpopulations**

Input: k transformations $A_1, A_2, \dots, A_k$; Training/Validation splits of m groups
Require: Number of iterations S, SGD steps α, learning rate η
Output: $(Q_1, w_1^{(S)}), (Q_2, w_2^{(S)}), \dots, (Q_m, w_m^{(S)})$; $\theta^{(S)}$

```
1: for $g = 1, \dots, m$ do
2:     Compute an augmentation scheme $Q_g$ for a specific group $\hat{P}_g$ using Algorithm 1
3: end for
4: Initialize model parameters $\theta^{(0)}$; Set weight variables $w^{(0)}$ as the uniform proportions $[1/m, \dots, 1/m]$
5: for $t = 0, \dots, S-1$ do
6:     Update $\theta^{(t)}$ with $\alpha$ SGD steps to get $\theta^{(t+1)}$
7:     Update $w^{(t+1)}$ from $w^{(t)}$ according to equations (5) and (6)
8: end for
9: return $(Q_1, w_1^{(S)}), (Q_2, w_2^{(S)}), \dots, (Q_m, w_m^{(S)})$; $\theta^{(S)}$
```

3.2.1 Running Time Analysis

Next, we examine the runtime of the weighted training algorithm. Compared to the standard SGD algorithm, our algorithm includes updating the group weights using equations (5) and (6). This step involves computing the inverse of the Hessian matrix on the average loss and calculating the product between the inverse of the Hessian matrix with the gradient vector of each group's loss. To avoid explicitly computing H^{-1}, we use the following method that:

- First, estimate the implicit Hessian-vector product $s := H^{-1}\big(\sum_{g=1}^{m} q_g \nabla_\theta \hat{L}_g(f_{\theta^w})\big)$. We use the conjugate gradient method with stochastic estimation. This method uniformly samples n data points $\{(x_j, y_j)\}_{j=1}^{n}$, recursively computes $H_j^{-1}v = v + (I - \nabla_\theta^2 L(f_\theta(x_j), y_j))H_{j-1}^{-1}v$, and takes $H_n^{-1}v$ as the final estimate of $H^{-1}v$. For n data points and $\theta \in \mathbb{R}^p$, this method takes $O(np)$ time.
- Secondly, compute $d_i = -s^\top \nabla_\theta \hat{L}_i(f_{\theta^w}; Q_i)$ as an inner product between s and the gradient on each group's loss. Computing d_i for all groups $i = 1, \dots, m$ takes $O(mp)$ time.

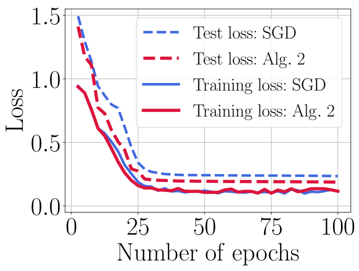
Taken together, updating the group weights takes $O(np + mp)$ time. In our implementation, we conduct the above estimation using one batch of b data points from each group, leading to a running time of $O(bmp + mp) = O(bmp)$.

Empirically, we find that our algorithm takes time comparable to SGD. To illustrate, we train a three-layer graph neural network on the protein graph classification data set. Our algorithm takes 1.8 GPU hours, which is only 1.1× of SGD (which takes 1.6 GPU hours). In training ResNet-50 on a wildlife image classification data set, our algorithm takes 0.8 GPU hours, which is only 1.3× of SGD (which takes 0.6 hours).

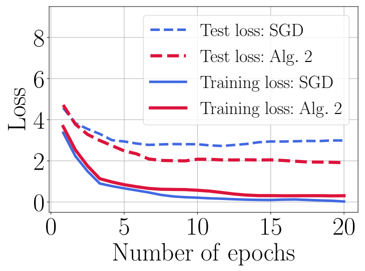
Furthermore, our algorithm converges in a comparable number of iterations as SGD. As illustrated in Figure 6, on both data sets, the training loss of our algorithm converges similarly to SGD, while our algorithm achieves a lower loss on a test set.

4 Experiments

In this section, we evaluate the performance of Algorithm 2 for various data sets in terms of both the runtime and the performance of the augmentation scheme, evaluated on a downstream task. We will evaluate in both supervised and self-supervised learning settings. In particular, we will compare our method with existing optimization methods for searching augmentations. We will also compare against augmentation schemes



(a) Training a three-layer NN for graph classification



(b) Training a ResNet-50 for image classification

Figure 6: We illustrate that Algorithm 2 converges in a comparable number of iterations as SGD. The left figure shows the results of training a three-layer neural network on the protein graph classification data set. The right figures show the result of training a ResNet-50 on an image classification data set.

designed based on domain knowledge and its variants, such as RandAugment and SimCLR. We summarize a few key findings from our experiments.

- First, we evaluate our algorithm in supervised learning settings, considering both graph data sets (which we collect from AlphaFold APIs) and image data sets.

  We find that our algorithm can outperform recent augmentation search methods on the graph classification data set by **1.9**% with **43**% less runtime and on the image classification data set by **3.0**% with **38**% less GPU hours.

  Compared to RandAugment (which does not search for a composition scheme), we can achieve **4.3**% and **5.7**% better results on the graph and image data sets, respectively.

- Next, we apply our algorithm to select an augmentation scheme in contrastive learning settings.

  Compared with existing augmentation search methods, we can now get up to **7.4%** improvement across eight graph and image data sets. The runtime is also reduced by **32**%.

  Compared to SimCLR, as illustrated in Figure 3, the augmentation scheme discovered by our algorithm can deliver comparable performance with SimCLR on CIFAR-10. On a medical image classification task, we now find another scheme that can outperform the SimCLR by **5.9**%.

  The improvement of our algorithm over the baselines is statistically significant. We conduct the Wilcoxon signed-rank test on the performances of our algorithm and the baselines over the ten data sets. Comparing our algorithm with the recent augmentation search method and with RandAugment, the test has a p-value of 0.0025 and 0.0019, respectively. This shows that the confidence score for exhibiting no improvement would be less than 1%.

- Lastly, we provide ablation studies to justify the necessity of both parts (finding the composition for each group and the weighted combination) of the algorithm for achieving the final performance.

We have provided our implementation in the online repository: `https://anonymous.4open.science/r/Learning-Tree-Structured-Composition-of-Data-Augmentation`. One can directly load our newly constructed data set in the repository. First, download the dataset in this link `https://drive.google.com/file/d/1Bbl-urCGbnUNim6A0ddobI8MMxeiW9Bu/view?usp=sharing`. Then, place it under the protein function classification folder and directly run the training script. For more details, please see the online repository.

4.1 Experimental Setup

We apply our algorithm to two settings. The first setting is supervised learning, which uses data augmentation to generate additional labeled examples. The second setting is contrastive learning, which uses data augmentation to generate contrastive examples.

Data sets. For supervised learning settings, we first construct a new graph classification data set for protein function prediction collected from the AlphaFold Protein Structure Database,[1]. This data set includes 20,504 human proteins structured as undirected graphs, where nodes correspond to amino acid residues labeled with one of 20 amino acid types. The edges are the spatial distances between two amino acids thresholded at 6Å between two Cα atoms. The task is a multilabel classification problem containing 1,198 protein functions, each function viewed as a binary label. We split the data set into 16 groups by different intervals of graph sizes and average degrees. The idea is to group graphs of similar sizes together. To determine the number of groups, we vary the number of groups between 4, 9, 16, and 25, which correspond to split graph sizes and average degrees into 2, 3, 4, and 5 intervals. We observe that splitting the groups further beyond 16 brings no obvious increase in the validation performance. Thus, we chose 16 groups.

The main difference between our data set and previous data sets (Borgwardt et al., 2005; Hu et al., 2020a) is the coverage of protein functions. The data set from Borgwardt et al. (2005) predicts whether a protein is an enzyme as a binary classification task. The data set from Hu et al. (2020a) classifies a protein into 37 taxonomic groups as different species as a multiclass classification problem. Our data set involves 1,198 protein functions using Gene Ontology (GO) annotation. Another difference is in constructing the graph structure. Borgwardt et al. (2005) and our data set abstract the structure of the protein itself as a graph. Hu et al. (2020a) constructs the graph based on a relationship between proteins in protein association networks. The third difference is that the graphs in our data set exhibit more significant variations in their graph sizes, ranging from 16 to 34,350. The comparison with the two previous data sets is summarized in Table 3.

Table 3: Comparison of our protein graph classification data set to two existing data sets. Our data set is constructed from a newer data source and covers a broader range of (protein) functions than previous ones.

	PROTEINS	OGBG-PPA	Our Data Set
# Graphs	1113	158,100	20,504
# Functions	2	37	1,198
# Nodes	Between $4 \sim 620$	Between $50 \sim 300$	Between $16 \sim 34,350$
Average Degree	Between $3.4 \sim 10.1$	Between $2.0 \sim 240.9$	Between $1.8 \sim 9.4$
Prediction Task	A binary classification task of whether or not the protein is an enzyme.	Multiclass classification task for 37 taxonomic groups as species, e.g., mammals, bacterial families, archaeans.	Multilabel classification task for 1198 GO terms that describe the biological function of proteins.
Types of Graphs	Attributed and undirected graphs, where nodes represent secondary structure elements (SSE), and edges represent spatial closeness.	Undirected graphs of protein associations, where nodes represent proteins and edges indicate biologically meaningful associations between proteins.	Undirected graphs, where nodes represent the amino acids and edges represent their spatial distance thresholded at 6Å between two Cα atoms.
Data Source	Protein data bank	Protein association network	AlphaFold APIs

Next, we consider an image classification task using the iWildCam data set from the WILDS benchmark (Beery et al., 2021; Koh et al., 2021), where images are collected from different camera traps with varied illumination, camera angles, and backgrounds. The task is multiclass classification: given a camera trap image, predict the labels as one of 182 animal species. We evaluate the macro-F_1 score on this data set, which is the average F_1 score over every class. We consider three groups from three cameras with the most significant number of images in the data set.

For contrastive learning settings, we consider image classification data sets for contrastive learning, including CIFAR-10 and a medical image data set containing eye fundus images for diabetic retinopathy classifications. The task on the medical images is multiclass classification: given an eye fundus image, predict the severity of diabetic retinopathy into five levels. We consider three groups as three hospitals, which differ in the patient

[1]https://alphafold.ebi.ac.uk/

population and imaging devices. The sources for the three subsets of medical images of the eye fundus are available in the following: Messidor,[2] APTOS,[3] and Jinchi[4].

Then, we consider six graph classification data sets from TUdatasets (Morris et al., 2020), including NCI1, Proteins, DD, COLLAB, REDDIT, and IMDB. The first three data sets contain small molecular graphs where nodes represent atoms and edges represent chemical bonds. The task is to predict the biological properties of molecular graphs. The latter three data sets contain social networks abstracted from scientific collaborations, online blogs, and actor collaborations. The task is to predict the topic of the social network. We split each data set into four groups by graph sizes and average degrees.

Baselines. We compare our approach with pre-specified data augmentation methods, including ones designed for specific group shifts, and optimization methods that search for compositions. For supervised learning settings, we consider RandAugment (Cubuk et al., 2020) and methods for group shifts, including Learning Invariant Predictors with Selective Augmentation (LISA) (Yao et al., 2022) and targeted augmentations (Gao et al., 2023). We also consider optimization methods, including GraphAug (Luo et al., 2023) on graph data sets and Stochastic Compositional Augmentation Learning (SCALE) (Chatzipantazis et al., 2023) on image data sets. We report the performance of ERM to show relative performance.

For contrastive learning settings, we consider pre-specified augmentation methods, including SimCLR (Chen et al., 2020b) and LISA (Yao et al., 2022) on image data sets, and InfoGraph (Sun et al., 2020), RandAugment (Cubuk et al., 2020), GraphCL (You et al., 2020) on graph data sets. We consider optimization methods, Joint Augmentation Optimization (JOAO) (You et al., 2021) on graph data sets, and SCALE (Chatzipantazis et al., 2023) on image data sets. We also include the performance of using pretrained features to show relative performance on image data sets.

Implementations. We use four graph augmentation methods for graph data sets as in You et al. (2020), including DropNodes, PermuteEdges, Subgraph, and MaskingNodes. Each method modifies a fraction of nodes, edges, or node features. Specifically, DropNodes deletes a randomly sampled set of nodes and their connections. PermuteEdges adds and deletes a randomly sampled set of edges. Subgraph extracts a subgraph generated by random walks from randomly sampled nodes in the graph. NodeMasking masks a randomly sampled fraction of node attributes to zero. We consider five fractions of modification uniformly spaced between 0 and 0.5 for each method as their perturbation magnitudes. In total, there are $k = 20$ options.

For image data sets, we use sixteen image augmentation methods as in Cubuk et al. (2019), including ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, and Sample Pairing. We consider five values of perturbation magnitude uniformly spaced between intervals described in Cubuk et al. (2019). In total, there are $k = 80$ options.

We use a three-layer graph neural network on graph data sets from Hu et al. (2020b). We use pretrained ResNet-50 on image data sets as in Koh et al. (2021).

In terms of hyperparameters, we search the augmentation tree of maximum depth $d = 4$ and ten probabilities for applying each augmentation uniformly spaced between 0 and 1.0. Searching with more depth and probability values does not improve performance. For the weighted training algorithm, we tune the learning rate η between $0.01, 0.1, 1.0$ and the SGD steps α between $25, 50, 100$.

4.2 Experimental Results

Supervised learning settings. We first discuss the results in supervised learning settings, as reported in Table 4. We compare our algorithm with two types of baselines, including pre-specified augmentation methods and optimization methods that search for the composition.

On the protein graph classification data set, our algorithm outperforms RandAugment, which randomly samples a composition of augmentations for each batch of data, and LISA, which applies the mixup technique

[2]https://www.adcis.net
[3]https://kaggle.com/competitions/aptos2019
[4]https://figshare.com

Table 4: We compare our algorithm with several existing data augmentation schemes on a graph classification data set (left) and an image classification data set (right). In particular, the left-hand side shows the average test AUROC scores for protein function prediction. The right shows the test macro-F_1 score on the image classification data set. We report the averaged results over five random seeds.

	Graph Classification	Image Classification
Training Set Size	12,302	6,568
Validation Set Size	4,100	426
Testing Set Size	4,102	789
# Classes	1,198	182
Metric	AUROC	Macro-F_1
ERM	71.3 $\pm$ 0.3	52.4 $\pm$ 1.1
RandAugment	71.8 $\pm$ 0.8	58.9 $\pm$ 0.4
LISA	73.2 $\pm$ 0.3	59.7 $\pm$ 0.3
Targeted Augmentation	-	56.3 $\pm$ 0.4
GraphAug	73.5 $\pm$ 0.3	-
SCALE	-	60.4 $\pm$ 0.5
Algorithm 2 (Ours)	**74.9** $\pm$ **0.4**	**62.3** $\pm$ **0.6**

between input examples, by **4.3**% and **2.3%**, respectively. This shows the benefit of searching augmentations over specifying a data augmentation beforehand. Moreover, our algorithm improves over the previous optimization method, GraphAug, by **1.9**%.

On the image classification data set, our algorithm outperforms pre-specified data augmentation methods, including RandAugment, targeted augmentation, and LISA, by **6.8%** on average. Our algorithm also improves over the optimization method, SCALE, by **3.0**%.

We report the runtime of our algorithm. Recall that our algorithm first finds augmentation composition in each group and then learns a weighted combination of them. Our algorithm takes 8.2 hours on the graph data set and 2.6 hours on the image data set. This takes **44**% and **38**% less than the optimization methods on the graph and image data sets, which use 14.4 and 4.2 hours, respectively. Moreover, as discussed in Section 3.2, the weighted training procedure of our algorithm takes a comparable runtime as methods that do not search for the composition.

Contrastive learning settings. Next, we extend our algorithm to contrastive learning settings. Again, we consider pre-specified augmentation methods, such as SimCLR, and existing optimization methods. We evaluate trained models by linear evaluation using an SVM classifier on the contrastive features.

We first report the results of contrastive learning on image data sets in Table 5. On CIFAR-10, which contains natural images, our algorithm performs on par with SimCLR designed by domain experts. On a medical image data set, we observe that the SimCLR augmentation scheme does not work on medical images and even decreases the performance compared to just using the pretrained network's features. By finding compositions of augmentations, our algorithm improves over SimCLR by **5.9**%. In both data sets, our algorithm outperforms RandAugment by **2.4**% on average.

Moreover, our algorithm improves an optimization method, SCALE, by **1.1**% on both image data sets on average. Our algorithm also takes **32**% less runtime than the optimization method.

Then, we apply our algorithm to graph contrastive learning. We evaluate the performance by 10-fold cross-validation as in You et al. (2020). Results are reported in Table 6. Compared to pre-specified augmentation methods, our algorithm outperforms RandAugment by up to **20**% and GraphCL by up to **17**% across six data sets. Compared to the optimization method, JOAO, our algorithm can improve upon the method by up to **7.1%**. Our algorithm benefits from splitting graphs into groups with similar sizes and degrees before

Table 5: We compare our algorithm with several augmentation methods in contrastive learning on image data sets. We report the test accuracy on CIFAR-10 and a medical image classification data set. We conduct contrastive learning using the ImageNet-pretrained ResNet-50 and evaluate the test accuracy using linear evaluations of the network's last-layer features. Results are averaged over five random seeds.

	Natural Image Classification	Medical Image Classification
Training Set Size	45,000	10,659
Validation Set Size	5,000	2,670
Test Set Size	5,000	3,072
# Classes	10	5
ImageNet Pretrained Features	88.9% $\pm$ 0.0	70.1% $\pm$ 0.0
SimCLR	**92.8**% $\pm$ 0.5	69.1% $\pm$ 1.1
RandAugment	89.8% $\pm$ 0.7	72.4% $\pm$ 0.5
LISA	-	72.1% $\pm$ 0.5
SCALE	91.3% $\pm$ 0.3	72.4% $\pm$ 0.2
Algorithm 2 (Ours)	**93.0**% $\pm$ 0.4	**73.2**% $\pm$ 0.3

Table 6: We compare our algorithm with several existing augmentation methods in graph contrastive learning. We report the test classification accuracy on several graph prediction data sets. We first conduct contrastive learning using three-layer graph neural networks and evaluate the test accuracy using linear evaluations of the network's last-layer features. Results are averaged over ten-fold cross-validation.

	NCI1	PROTEINS	DD	COLLAB	REDDIT-B	IMDB-B
# Graphs	4,110	1,113	1,178	5,000	2,000	1,000
# Classes	2	2	2	3	2	2
InfoGraph	76.2%	74.4%	72.8%	70.6%	82.5%	73.0%
RandAugment	62.0%	72.2%	75.7%	58.1%	76.3%	55.2%
GraphCL	77.8%	74.3%	78.6%	71.3%	**89.4**%	71.1%
JOAO	78.0%	74.5%	77.4%	69.5%	86.4%	70.8%
Algorithm 2 (Ours)	**79.3**%	**77.7**%	**78.9**%	**78.4**%	**89.4**%	**74.4**%

applying graph contrastive learning. It is also plausible to use the augmentation scheme from GraphCL or JOAO and leverage our algorithm to combine the loss of each group.

Ablation studies. Lastly, we show that both parts of our algorithm contribute to achieving the results, including finding one augmentation composition for each group and aggregation by weighted training. We remove each part at one time and compare the performance before and after the removal. First, we remove finding one augmentation per group and replace it with a single augmentation on all groups. This resulted in a performance decrease of 0.6% and 1.0% on the protein graph and image classification data set, respectively. Second, we remove bilevel optimization and replace it with the uniform weighting of group losses. This leads to a performance decrease of 1.0% and 1.6% on the two data sets, respectively. Moreover, we analyze the trained model's features between groups. Our algorithm results in features that are more similar between groups than those used in a single augmentation composition. See Appendix B for the details.

Discussion concerning hyper-parameters. We discuss the impact of varying the hyper-parameters in our algorithms. Recall in Algorithm 1, we require the maximum depth of the tree-structured composition d and the number of probabilities $|H|$ between 0 and 1. In Algorithm 2, we require the number of SGD steps α, and learning rate η for updating combination weights. In each ablation study, we vary one hyper-parameter and keep the others unchanged. The fixed hyper-parameters are used as follows: depth of the tree as 4, number of probabilities as 10, SGD steps as 50, and learning rate as 0.1.

Table 7 reports the validation performance when varying the hyper-parameters in the supervised learning settings on the graph and image classification data sets. We notice that using the maximum depth as 4 and the number of probabilities H as 10 leads to the best results. Further increasing the depth and probability values brings no obvious improvement. Moreover, using the SGD steps as 50 and the learning rate as 0.1 leads to the best results for Algorithm 2. We also find that these hyper-parameters are useful for other settings. Thus, we use these values as the default parameters in the experiments.

Table 7: Ablation study of varying maximum depth of the tree d, the number of probability values $|H|$, the SGD steps α, and the learning rate η for updating the weights in Algorithm 2. We report the average validation AUROC scores on the protein graph classification data set and validation macro-F_1 score on the wildlife image classification data sets. The results are averaged over five random seeds.

	Graph classification (Val AUROC)			Image classification (Val F_1)		
d	2	3	4	2	3	4
Avg & Stdev	72.4 ± 0.1	74.2 ± 0.2	**74.5** ± 0.4	61.6 ± 0.6	65.0 ± 0.3	**65.3** ± 0.6
$\|H\|$	5	10	20	5	10	20
Avg & Stdev	72.9 ± 0.2	**74.5** ± 0.4	**74.5** ± 0.6	64.7 ± 0.3	**65.3** ± 0.6	**65.3** ± 0.4
α	25	50	100	25	50	100
Avg & Stdev	74.0 ± 0.2	**74.5** ± 0.4	73.4 ± 0.8	64.8 ± 0.3	**65.3** ± 0.6	64.2 ± 0.4
η	1.0	0.1	0.01	1.0	0.1	0.01
Avg & Stdev	71.0 ± 0.1	**74.5** ± 0.4	72.4 ± 0.8	60.5 ± 0.7	**65.3** ± 0.6	64.3 ± 0.5

4.3 Extension and Discussion

Extension. We can also apply the tree construction to semi-supervised learning, where we use data augmentation to perform consistency regularization on unlabeled examples as in UDA (Xie et al., 2020). We consider the CIFAR-10 and SVHN data sets. Following the setting of UDA, for CIFAR-10, we use 1,000 labeled examples and 49,000 unlabeled examples. For SVHN, we use 1,000 labeled examples and 64,932 unlabeled examples. We train randomly initialized Wide-ResNet-28-10 on both data sets, using SGD with a learning rate of 0.03 and 100,000 gradient update steps. To find a composition of data augmentation, we consider the same 16 types of image transformations used in other image data sets.

Table 8 reports the test error rates on both data sets. On both data sets, our algorithm outperforms UDA (which uses RandAugment) by 0.5%.

Table 8: We compare our algorithm with UDA, which uses RandAugment to perform consistency regularization in semi-supervised learning. We train randomly initialized WideResNet-28-10 on CIFAR-10 and SVHN data sets, each with 1,000 labeled examples. We report error rates on the test set, averaged over five random seeds.

	CIFAR-10 Error rates (%)	SVHN Error rates (%)
UDA (Xie et al., 2020)	4.9 ± 0.1	2.7 ± 0.1
Algorithm 1 (Ours)	**4.5** ± 0.1	**2.2** ± 0.1

Discussion. One justification for our empirical findings is that the trees generalize sequential augmentations, which have been the focus of prior work. Since we are searching inside this family of generalized structures, our results are expected to be on par (if not better) than the prior results. Again, we have to emphasize that much of the results in this literature are empirical, so establishing why our (or any other) method works well would be an interesting question for further research.

5 Conclusion

This paper designs an algorithm for learning the tree-structured composition of data augmentation. The algorithm uses a top-down recursive search method to find a tree with reduced running time complexity. Experiments validate that the algorithm reduces the runtime compared to existing search methods without decreasing downstream performance. Next, the algorithm is extended to the case of heterogeneous subpopulations. The algorithm first finds one tree for each group to account for heterogeneous features and then reweights each tree into a forest of trees. Extensive experiments show the empirical benefits over existing augmentation methods. The algorithm can be readily extended to tackle different loss metrics such as the worst-group loss. It may also be worth revisiting out-of-domain generalization where heterogeneous domains are present through the design of automatic data augmentation.

References

Micah Adler and Brent Heeringa. Approximating optimal binary decision trees. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pp. 1–9. Springer, 2008. 4

Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P. Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P. Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–29, 2000. 3

Sara Beery, Arushi Agarwal, Elijah Cole, and Vighnesh Birodkar. The iwildcam 2021 competition dataset. *CVPR*, 2021. 13

Gregory Benton, Marc Finzi, Pavel Izmailov, and Andrew G Wilson. Learning invariances in neural networks from training data. *NeurIPS*, 2020. 3

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 2005. 3, 13

Davide Buffelli, Pietro Liò, and Fabio Vandin. Sizeshiftreg: a regularization method for improving size-generalization in graph neural networks. *NeurIPS*, 2022. 3

Evangelos Chatzipantazis, Stefanos Pertigkiozoglou, Kostas Daniilidis, and Edgar Dobriban. Learning augmentation distributions using transformed risk minimization. *TMLR*, 2023. 5, 7, 14

Shuxiao Chen, Edgar Dobriban, and Jane H Lee. A group-theoretic framework for data augmentation. *The Journal of Machine Learning Research*, 21(1):9885–9955, 2020a. 4

Shuxiao Chen, Koby Crammer, Hangfeng He, Dan Roth, and Weijie J Su. Weighted training for cross-task learning. *ICLR*, 2022. 11, 23

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020b. 1, 3, 5, 7, 14

Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. *ICLR*, 2022. 3

Wyatt T Clark and Predrag Radivojac. Analysis of protein function and its prediction from amino acid sequence. *Proteins: Structure, Function, and Bioinformatics*, 2011. 2

Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *CVPR*, 2019. 1, 2, 5, 7, 14

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical data augmentation with no separate search. *CVPRW*, 2020. 14

Irena Gao, Shiori Sagawa, Pang Wei Koh, Tatsunori Hashimoto, and Percy Liang. Out-of-domain robustness via targeted augmentations. *ICML*, 2023. 2, 14

Anupam Gupta, Viswanath Nagarajan, and R Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. *Mathematics of Operations Research*, 42(3):876–896, 2017. 4

Steve Hanneke and Samory Kpotufe. On the value of target data in transfer learning. *NeurIPS*, 2019. 11, 23

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. *ECCV*, 2020. 3

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Meta approach to data augmentation optimization. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022. 3

Ignacio Hounie, Luiz FO Chamon, and Alejandro Ribeiro. Automatic data augmentation via invariance-constrained learning. In *ICML*, 2023. 3

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020a. 3, 13

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *ICLR*, 2020b. 8, 14

John Jumper, Richard. Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zidek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. 3

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *ICML*, 2021. 13, 14

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In *NeurIPS*, 2019. 1, 5, 7

Chen Lin, Minghao Guo, Chuming Li, Xin Yuan, Wei Wu, Junjie Yan, Dahua Lin, and Wanli Ouyang. Online hyper-parameter learning for auto-augmentation strategy. In *Proceedings of the IEEE/CVF international conference on computer vision*, 2019. 3

Youzhi Luo, Michael McThrow, Wing Yee Au, Tao Komikado, Kanji Uchino, Koji Maruhashi, and Shuiwang Ji. Automated data augmentations for graph classification. *ICLR*, 2023. 2, 3, 8, 14

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020. 14

Predrag Radivojac. Advancing remote homology detection: a step toward understanding and accurately predicting protein function. *Cell Systems*, 2022. 2

Alexander J Ratner, Henry Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. *NeurIPS*, 2017. 1

Burkhard Rost, Predrag Radivojac, and Yana Bromberg. Protein function in precision medicine: deep understanding with machine learning. *FEBS Letters*, 2016. 2

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *ICLR*, 2020. 4

Han Shao, Omar Montasser, and Avrim Blum. A theory of pac learnability under transformation invariances. *Advances in Neural Information Processing Systems*, 35:13989–14001, 2022. 4

Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *ICLR*, 2020. 14

Sen Wu, Hongyang R Zhang, Gregory Valiant, and Christopher Ré. On the generalization effects of linear transformations in data augmentation. In *ICML*, 2020. 3, 4

Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *NeurIPS*, 2020. 1, 3, 17

Kaiwen Yang, Yanchao Sun, Jiahao Su, Fengxiang He, Xinmei Tian, Furong Huang, Tianyi Zhou, and Dacheng Tao. Adversarial auto-augment with label preservation: A representation learning principle guided approach. *NeurIPS*, 2022. 3

Shuo Yang, Yijun Dong, Rachel Ward, Inderjit S Dhillon, Sujay Sanghavi, and Qi Lei. Sample efficiency of data augmentation consistency regularization. In *AISTATS*, 2023. 3

Huaxiu Yao, Yu Wang, Sai Li, Linjun Zhang, Weixin Liang, James Zou, and Chelsea Finn. Improving out-of-distribution robustness via selective augmentation. In *ICML*, 2022. 14

Gilad Yehudai, Ethan Fetaya, Eli Meirom, Gal Chechik, and Haggai Maron. From local structures to size generalization in graph neural networks. In *ICML*, 2021. 3

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *NeurIPS*, 33, 2020. 3, 8, 14, 15

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In *ICML*, 2021. 14

Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In *ICLR*, 2020. 1

Yu Zheng, Zhi Zhang, Shen Yan, and Mi Zhang. Deep autoaugment. *ICML*, 2021. 3

A Derivation of Equation (5)

By regarding θ^w as an implicit function of w and applying the chain rule, we have the following:

$$\frac{\partial \hat{L}(f_{\theta^w})}{\partial w} = \left(\nabla_\theta \hat{L}(f_{\theta^w})\right)^\top \frac{\partial \theta^w}{\partial w}. \tag{7}$$

By the definition of θ^w, we must have that θ^w is a minimizer of $\sum_{g=1}^m w_g \cdot \hat{L}(f_\theta; Q_g)$. Thus, it satisfies that the gradient to θ is zero, given a weight w. In the following, we write the gradient as an implicit function:

$$F(w,\theta) = \nabla_\theta \sum_{g=1}^m w_g \cdot \hat{L}(f_\theta; Q_g) = \sum_{g=1}^m w_g \cdot \nabla_\theta \hat{L}(f_\theta; Q_g). \tag{8}$$

For a stationary point solution (w, θ^w) of equation (4), we have $F(w, \theta^w) = 0$. By the implicit function Theorem, if $F(w,\theta)$ is continuously differentiable and $\partial F(w,\theta)/\partial\theta$ is invertible. We can conclude that for a (w,θ) near some $(\tilde{w},\tilde{\theta})$ satisfying $F(\tilde{w},\tilde{\theta}) = 0$, there exists a function mapping $w \mapsto \theta^w$ which is continuous and continuously differentiable and the derivative of this mapping over w is given by:

$$\frac{\partial \theta^w}{\partial w} = -\left(\left.\frac{\partial F(w,\theta)}{\partial \theta}\right|_{\theta^w}\right)^{-1}\left(\left.\frac{\partial F(w,\theta)}{\partial w}\right|_{\theta^w}\right). \tag{9}$$

Replacing equation (9) into equation (7), we get

$$\frac{\partial \hat{L}(f_{\theta^w})}{\partial w_i} = -\left(\sum_{g=1}^m w_g \nabla_\theta \hat{L}(f_{\theta^w}; Q_g)\right)^\top \left(\sum_{g=1}^m w_g \nabla_\theta^2 \hat{L}(f_{\theta^w}; Q_g)\right)^{-1} \nabla_\theta \hat{L}(f_{\theta^w}; Q_i). \tag{10}$$

B Feature Similarity Analysis

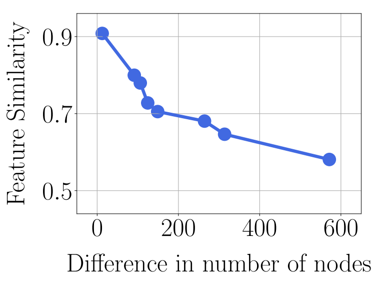
In this section, we explore how our algorithm affects the learned features of different groups. We use the protein graph dataset as an example. To quantify the difference between groups, we compute a similarity score $s(i,j)$ between the last-layer features' covariance matrices of groups i and j. For group i, denote $X_i \in \mathbb{R}^{n_i \times d}$ as the feature vectors of n_i samples with dimension d. Denote the covariance matrix as $X_i^\top X_i$. We use the rank-r_i approximation to the covariance matrix $U_{i,r_i} D_{i,r_i} U_{i,r_i}^\top$, where r_i is chosen to contain 99% of the singular values. Then, we measure the similarity as

$$s(i,j) = \frac{\|(U_{i,r_i} D_{i,r_i}^{1/2})^\top U_{j,r_j} D_{j,r_j}^{1/2}\|_F}{\|U_{i,r_i} D_{i,r_i}^{1/2}\|_F \|U_{j,r_j} D_{j,r_j}^{1/2}\|_F}. \tag{11}$$

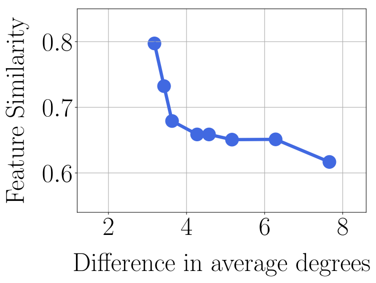
Higher $s(i,j)$ indicates greater similarity.

First, we confirm that the graphs of various graph sizes or average degrees exhibit distinct features. Recall that we split groups based on graph sizes and average degrees. For every two groups, we train a model on their combined dataset and measure the feature similarity score between them. At the same time, we compute the differences in average graph sizes and average degrees between the two groups. As shown in Figure 7a, the feature similarity between the two groups drops as the disparity between their graph sizes grows. In Figure 7b, similar results are found when we measure the difference in average degrees.

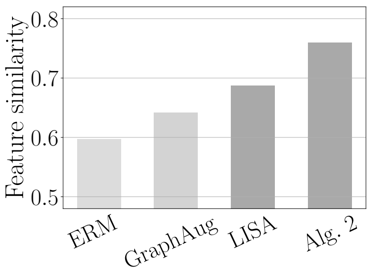
Second, our method can improve the feature similarity between the two groups. As shown in Figure 7c, we find that using augmentation leads to a higher feature similarity score, averaged over every pair of groups, than ERM. By finding an augmentation composition in each group, our algorithm further improves the feature similarity score by **7.3%** over using a single augmentation.



(a) Groups of varied graph sizes



(b) Groups of varied average degrees



(c) Average feature similarity

Figure 7: We observe that groups with larger differences in graph size or average degree exhibit more distinct features in a trained model. Using augmentation can result in more similar hidden features between groups than ERM. With augmentations in each group and weighted training, our algorithm further improves the similarity between features of different groups by 7.1% over using a single augmentation composition.

C Consistency of Bilevel Optimization

This section derives a generalization bound for the bilevel optimization algorithm. The result shows that when the number of samples goes to infinity, the gap between training and test errors will shrink to zero, plus a discrepancy distance term between the source and target tasks. Let $\{P_g\}_{g=1}^m$ denote the m groups of augmented data distributions. For group g, we have n_g i.i.d samples $\hat{P}_g = (x_{g,i}, y_{g,i})_{i=1}^{n_g}$ from P_g. We denote P_0 as a target data distribution. Given a bounded loss function $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto [0,1]$. Let $L_g(\theta)$ and $\hat{L}_g(\theta)$ denote the expected and empirical loss on group g, respectively. We denote the minimizer of the weighted empirical loss as

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} \sum_{g=1}^{m} w_g \hat{L}_g(\theta). \tag{12}$$

We denote the optimal representation as:

$$\theta_0^\star \in \arg\min_{\theta_0 \in \Theta} L_0(\theta_0).$$

We denote the w-weighted optimal representation as:

$$\bar{\theta}^w \in \arg\min_{\theta \in \Theta} \sum_{g=1}^{m} w_g L_g(\theta).$$

Since $\bar{\theta}^w$ may not be unique, we introduce the function space $\bar{\theta}^w \in \Theta$ to include all w-weighted $\bar{\theta}^w$.

Definition C.1 ((ρ, C_ρ)-transferable)**.** *A representation $\theta \in \Theta$ is (ρ, C_ρ)-transferable from w-weighted source subsets to the target subset, if there exists $\rho > 0, C_\rho > 0$ such that for any $\bar{\theta}^w \in \bar{\Theta}^w$, we have*

$$L_0(\theta) - L_0(\bar{\theta}^w) \le C_\rho \Big(\sum_{g=1}^{m} w_g \left(L_g(\theta) - L_g(\bar{\theta}^w) \right) \Big)^{\frac{1}{\rho}}.$$

Next, we state two technical assumptions. The first assumption, commonly used in the literature, describes the loss function's Lipschitz continuity.

Assumption C.1. *Let the loss function $\ell : \mathcal{X} \times \mathcal{Y} \to [0,1]$ be C-Lipschitz in $x \in \mathcal{X}$, meaning for all $x_1, x_2 \in \mathcal{X}$ and $y \in \mathcal{Y}$, the following holds*

$$|\ell(x_1, y) - \ell(x_2, y)| \le C \cdot \|x_1 - x_2\|.$$

The second assumption describes the covering size of the functional class Θ.

Assumption C.2. *There exist constants C_Θ and v_Θ greater than 0 such that for any probability measure $\mathbb{Q}_\mathcal{X}$ on $\mathcal{X} \subseteq \mathbb{R}^d$, we have*

$$\mathcal{N}\Big(\epsilon; \Phi; L^2(\mathbb{Q}_\mathcal{X})\Big) \leq \left(\frac{C_\Theta}{\epsilon}\right)^{v_\Theta} \text{ for any } \epsilon > 0,$$

where $\mathcal{N}\big(\epsilon; \Phi; L^2(\mathbb{Q}_\mathcal{X})\big)$ is the minimum number of $L^2(\mathbb{Q}_\mathcal{X})$ balls with radius ϵ required to cover Θ. Here, the $L^2(\mathbb{Q}_\mathcal{X})$ distance between two vector-valued functions θ and ψ in Θ is defined as

$$L^2(\mathbb{Q}_\mathcal{X}) = \left(\int \|\theta(x) - \psi(x)\|^2 \, d\mathbb{Q}_\mathcal{X}(x)\right)^{\frac{1}{2}}.$$

Given some weight vector w, let us define

$$\operatorname{dist}\Big(\sum_{g=1}^m w_g P_g, P_0\Big) := \underset{\bar\theta^w \in \bar\Theta^w}{sup} L_0(\bar\theta^w) - L_0(\theta_0^\star),$$

which represents the distance between the source w-weighted and target groups. Now, we are ready to state the main result of this section.

Theorem C.3. *Let $\hat\theta$, $\bar\theta^w$, $\theta_0^\star$ be defined as the above equations with a fixed w. Suppose Assumptions C.1 and C.2 are true. Let $\delta \in (0,1)$ be a fixed real number. Then, for any representation $\hat\theta \in \Theta$ where $\hat\theta$ is (ρ, C_ρ)-transferable, with probability at least $1-\delta$, we have:*

$$L_0(\hat\theta) - L_0(\theta_0^\star) \leq \operatorname{dist}\left(\sum_{g=1}^m w_g P_g, P_0\right) + C_\rho \left(\frac{C_1\sqrt{v_\Theta \log(C_\Theta C)} + C_2\sqrt{\log(\delta^{-1})}}{\sqrt{N_w}}\right)^{\frac{1}{\rho}},$$

where $N_w = \big(\sum_{g=1}^m \frac{w_g^2}{n_g}\big)^{-1}$, C_1 and C_2 are two constants that do not grow with the sample sizes.

At a high level, the first distance term represents the bias increase to transfer from a different data distribution, and the second term captures variance (or the error of learning imperfect representation $\bar\theta^w$). This term reduces under pooling, e.g., consider a scenario where n_g is the same across groups, then through N_w, the second term reduces by a factor of m^{-1}. Thus, this second term captures the benefit of knowledge transfer to the target group. The proof technique is based on carefully examining the bilevel optimization algorithm using covering numbers Chen et al. (2022); Hanneke & Kpotufe (2019).

Proof. Let Θ be the domain for which θ lies in. Consider a bounded loss function $\ell : \mathcal{X} \times \mathcal{Y} \mapsto [0,1]$. To establish the result, we begin by decomposing the difference between $L_0(\hat\theta)$ and $L_0(\bar\theta^w)$. Starting with the straightforward expression, we proceed with the following calculations:

$$\begin{aligned} L_0(\hat\theta) - L_0(\bar\theta^w) &= \underbrace{L_0(\hat\theta) - L_0(\bar\theta^w) + L_0(\bar\theta^w) - L_0(\theta_0^\star)}_{\text{adding and subtracting } L_0(\bar\theta^w)} \\ &\leq C_\rho \left(\sum_{g=1}^m \omega_g (L_g(\hat\theta) - L_g(\bar\theta^w))\right)^{\frac{1}{\rho}} + \underbrace{\sup_{\bar\theta^w \in \bar\Theta^w} \big(L_0(\bar\theta^w) - L_0(\theta_0^\star)\big)}_{\text{upper bounded by the supremum}} \qquad (13) \\ &= C_\rho \left(\sum_{g=1}^m \omega_g (L_g(\hat\theta) - L_g(\bar\theta^w))\right)^{\frac{1}{\rho}} + \operatorname{dist}\left(\sum_{g=1}^m \omega_g P_g, P_0\right). \end{aligned}$$

Now we focus on the first term from above, which is $\sum_{g=1}^m \omega_g(L_g(\hat{\theta}) - L_g(\bar{\theta}^w))$. We have

$$\begin{aligned}
\sum_{g=1}^m \omega_g(L_g(\hat{\theta}) - L_g(\bar{\theta}^w)) &= \sum_{g=1}^m \omega_g \left(L_g(\hat{\theta}) - \hat{L}_g(\hat{\theta}) + \hat{L}_g(\hat{\theta}) - \hat{L}_g(\bar{\theta}^w) + \hat{L}_g(\bar{\theta}^w) - L_g(\bar{\theta}^w)\right) \\
&\leq \sum_{g=1}^m \omega_g \left((L_g(\hat{\theta}) - \hat{L}_g(\hat{\theta}) + \hat{L}_g(\bar{\theta}^w) - L_g(\bar{\theta}^w)\right) \qquad (14) \\
&\leq \sup_{\theta\in\Theta} \left(\sum_{g=1}^m \omega_g \left((L_g(\theta) - \hat{L}_g(\theta) + \hat{L}_g(\bar{\theta}^w) - L_g(\bar{\theta}^w)\right)\right) \\
&= \sup_{\theta\in\Theta} \left(\sum_{g=1}^m \omega_g \cdot \frac{1}{n_g} \sum_{i=1}^{n_g} \left(L_g(\theta) - \ell(f_\theta(\mathbf{x}_{g,i}), y_{g,i}) + \ell(f_{\bar{\theta}^w}(\mathbf{x}_{g,i}), y_{g,i}) - L_g(\bar{\theta}^w)\right)\right).
\end{aligned}$$

Denote the last equation above as $G(\{\mathbf{z}_{g,i}\})$, where $\mathbf{z}_{g,i} = \{\mathbf{x}_{g,i}, y_{g,i}\}$. Step (14) is by noting that $\hat{L}_g(\hat{\theta}) - \hat{L}_g(\bar{\theta}^w)$ is at most zero according to equation (12).

Next, we will apply McDiarmid's inequality. Let us fix two indices $1 \leq g' \leq m$ and $1 \leq i_{g'} \leq n_g$. Let us define $\{\tilde{\mathbf{z}}_{g',i}\}$ by replacing $\mathbf{z}_{g',i_{g'}}$ with another $\tilde{\mathbf{z}}_{g',i_{g'}} = (\tilde{\mathbf{x}}_{g',i_{g'}}, \tilde{y}_{g',i_{g'}}) \in \mathcal{X} \times \mathcal{Y}$. Given that $\{\mathbf{z}_{g,i}\}$ and $\{\tilde{\mathbf{z}}_{g',i}\}$ differ by only one element, we have

$$\begin{aligned}
|G(\{\mathbf{z}_{g,i}\}) - G(\{\tilde{\mathbf{z}}_{g',i}\})| =& \Bigg| \sum_{g\neq g'} \sum_{i=1}^{n_g} \frac{\omega_g}{n_g} \left(L_g(\theta) - \ell(f_\theta(\mathbf{x}_{g,i}), y_{g,i}) + \ell(f_{\bar{\theta}^w}(\mathbf{x}_{g,i}), y_{g,i}) - L_g(\bar{\theta}^w)\right) \\
&+ \sum_{i\neq i_{g'}} \frac{\omega_{g'}}{n_{g'}} \left(L_{g'}(\theta) - \ell(f_\theta(\mathbf{x}_{g',i}), y_{g',i}) + \ell(f_{\bar{\theta}^w}(\mathbf{x}_{g',i}), y_{g',i}) - L_{g'}(\bar{\theta}^w)\right) \\
&+ \frac{\omega_{g'}}{n_{g'}} \left(L_{g'}(\theta) - \ell(f_\theta(\mathbf{x}_{g',i_{g'}}), y_{g',i_{g'}}) + \ell(f_{\bar{\theta}^w}(\mathbf{x}_{g',i_{g'}}), y_{g',i_{g'}}) - L_{g'}(\bar{\theta}^w)\right) \Bigg| \\
\leq& \frac{\omega_{g'}}{n_{g'}} \Bigg(\left|L_{g'}(\theta) - \ell(f_\theta(\mathbf{x}_{g',i_{g'}}), y_{g',i_{g'}})\right| + \left|L_{g'}(\bar{\theta}^w) - \ell(f_{\bar{\theta}^w}(\mathbf{x}_{g',i_{g'}}), y_{g',i_{g'}})\right| \\
&+ \left|L_{g'}(\theta) - \ell(f_\theta(\tilde{\mathbf{x}}_{g',i_{g'}}), \tilde{y}_{g',i_{g'}})\right| + \left|L_{g'}(\bar{\theta}^w) - \ell(f_{\bar{\theta}^w}(\tilde{\mathbf{x}}_{g',i_{g'}}), \tilde{y}_{g',i_{g'}})\right| \Bigg) \leq \frac{4\omega_{g'}}{n_{g'}}.
\end{aligned}$$

The last step above is based on the fact that the loss function is bounded between 0 and 1. Thus, we derive the following result:

$$\Pr\left(G(\{\mathbf{z}_{g,i}\}) - \mathbb{E}[G(\{\mathbf{z}_{g,i}\})] \geq \epsilon\right) \leq \exp\left(-\frac{2\epsilon^2}{\sum_{g=1}^m \sum_{i=1}^{n_g} \frac{16\omega_g^2}{n_g^2}}\right), \ \forall \epsilon > 0. \qquad (15)$$

Recall our previous definition that $N_w = \left(\sum_{g=1}^m \frac{\omega_g^2}{n_g}\right)^{-1}$. By setting δ as

$$\delta = \exp\left(-\frac{2\epsilon^2}{\sum_{g=1}^m \sum_{i=1}^{n_g} \frac{16\omega_g^2}{n_g^2}}\right),$$

we get that $\epsilon = 2\sqrt{2}\sqrt{\frac{\log(1/\delta)}{N_w}}$. Thus, Equation (15) can be equivalently stated as:

$$G(\{\mathbf{z}_{g,i}\}) \leq \mathbb{E}[G(\{\mathbf{z}_{g,i}\})] + 2\sqrt{2}\sqrt{\frac{\log(1/\delta)}{N_w}}, \qquad (16)$$

which holds with a probability of at least $1 - \delta$ for every $\delta \in (0, 1)$.

Next, for the function space Θ, let

$$M_\theta = \sqrt{N_w}\sum_{g=1}^{m}\sum_{i=1}^{n_g} r_{g,i}\frac{\omega_g}{n_g}\left(-\ell(f_\theta(\mathbf{x}_{g,i}), y_{g,i})\right),\ \forall\ \theta \in \Theta,$$

where $r_{g,i}$ are independent Rademacher random variables. For any $\theta_1, \theta_2 \in \Theta$, we define d as

$$d^2(\theta_1, \theta_2) = N_w \sum_{g=1}^{m}\sum_{i=1}^{n_g}\frac{\omega_g^2}{n_g^2}\left(\ell(f_{\theta_1}(\mathbf{x}_{g,i}), y_{g,i}) - \ell(f_{\theta_2}(\mathbf{x}_{g,i}), y_{g,i})\right)^2.$$

Now, we justify why this represents a random process with sub-Gaussian increments. In the definition of M_θ, the Rademacher random variable $r_{g,i}$ introduces randomness from the sign of each term. M_θ can thus be understood as an empirical average over random sign flips. Additionally, $d^2(\theta_1, \theta_2)$ computes the squared difference in losses under θ_1 and θ_2. The squared term ensures that this quantity is non-negative and gives a measure of "distance" between the two functions in terms of their losses. Combining both observations, we have that the difference $M_{\theta_1} - M_{\theta_2}$ is a random process with increments characterized by the metric d^2. Moreover,

$$\mathbb{E}\left[\exp\left(\lambda(M_{\theta_1} - M_{\theta_2})\right)\right] \le \exp\left(\frac{\lambda^2}{2}d^2(\theta_1, \theta_2)\right),\ \forall\ \lambda \ge 0, \theta_1 \in \Theta, \theta_2 \in \Theta.$$

This result shows that the tail probabilities of the process $M_{\theta_1} - M_{\theta_2}$ decay at least as fast as those of a Gaussian process, justifying the sub-Gaussian property.

Next, we use Dudley's entropy integral inequality conditioned on the randomness of $\mathbf{z}_{g,i}$ to obtain

$$\mathbb{E}\left[\sup_{\theta\in\Theta} M_\theta - M_{\bar{\theta}^w} | \{\mathbf{z}_{g,i}\}\right] \le 8\sqrt{2}\int_0^1 \sqrt{\log \mathcal{N}(\epsilon; \Theta; d)}\mathrm{d}\epsilon. \tag{17}$$

Recall the definition of the covering number from Assumption C.2. Given that

$$\sqrt{N_w}\mathbb{E}[G(\{\mathbf{z}_{g,i}\})] \le 2\sqrt{N_w} \times \mathbb{E}\left[\sup_{\theta\in\Theta}\sum_{g=1}^{m}\sum_{i=1}^{n_g} r_{g,i}\cdot\frac{\omega_g}{n_g}(\ell(f_{\bar{\theta}^w}(\mathbf{x}_{g,i}), y_{g,i}) - \ell(f_\theta(\mathbf{x}_{g,i}), y_{g,i}))\right],$$

Eq. (17) can be transformed to

$$\mathbb{E}\left[G(\{\mathbf{z}_{g,i}\})\right] \le \frac{16\sqrt{2}}{\sqrt{N_w}}\int_0^1 \sqrt{\log \mathcal{N}(\epsilon; \Theta; d)}\mathrm{d}\epsilon. \tag{18}$$

Next, we consider the covering number of Θ. We define

$$\mathbb{Q} = \sum_{g=1}^{m} N_w \frac{\omega_g^2}{n_g}\sum_{i=1}^{n_g}\frac{\delta_{\mathbf{x}_{g,i}}}{n_g},\ \text{where } \delta_{\mathbf{x}_{g,i}} \text{ represents a point mass at } \mathbf{x}_{g,i}.$$

Denote N_ϵ as $\mathcal{N}(\epsilon; \Theta; L^2(\mathbb{Q}))$. Let $\{\theta^{(1)}, \cdots, \theta^{(N_\epsilon)}\} \subseteq \Theta$ be an ϵ-covering of Θ with respect to $L^2(\mathbb{Q})$. This implies that for any $\theta \in \Theta$, there exists $j \in \{1, \cdots, N_\epsilon\}$ such that

$$\|f_\theta - f_{\theta^{(j)}}\|^2_{L^2(\mathbb{Q})} = \sum_{g=1}^{m}\sum_{i=1}^{n_g} N_\omega \frac{\omega_g^2}{n_g^2}\|f_\theta(\mathbf{x}_{g,i}) - f_{\theta^{(j)}}(\mathbf{x}_{g,i})\|^2 \le \epsilon^2.$$

In conclusion, from the previous steps, we have:

$$\begin{aligned} d^2(\theta, \theta^{(j)}) &= N_\omega \sum_{g=1}^{m}\sum_{i=1}^{n_g}\frac{\omega_g^2}{n_g^2}(\ell(f_\theta(\mathbf{x}_{g,i}), y_{g,i}) - l(f_{\theta^{(j)}}(\mathbf{x}_{g,i}), y_{g,i}))^2 \\ &\le C^2 N_\omega \sum_{g=1}^{m}\sum_{i=1}^{n_g}\frac{\omega_g^2}{n_g^2}\|f_\theta(\mathbf{x}_{g,i}) - f_{\theta^{(j)}}(\mathbf{x}_{g,i})\|^2 \\ &= C^2\|f_\theta - f_{\theta^{(j)}}\|^2_{L^2(\mathbb{Q})} \le C^2\epsilon^2. \end{aligned} \tag{19}$$

From the above, we have that

$$N(L_l\epsilon;\Theta;d) \le N_\epsilon \le \left(\frac{C_\Theta}{\epsilon}\right)^{v_\Theta} \Rightarrow \log N(\epsilon;\Theta;d) \le v_\Theta\left(\log(C_\Theta C) + \log\left(\frac{1}{\epsilon}\right)\right).$$

Applying the above to Eq. (18), we get

$$\begin{aligned}\mathbb{E}[G(\{\mathbf{z}_{g,i}\})] &\le \frac{16\sqrt{2}}{\sqrt{N_\omega}}\int_0^1 \sqrt{v_\Theta\left(\log(C_\Theta C) + \log\left(\frac{1}{\epsilon}\right)\right)}\,d\epsilon \\ &\le \frac{16\sqrt{2}\sqrt{v_\Theta \log(C_\Theta C)}}{\sqrt{N_\omega}}\left(1 + \int_0^1 \sqrt{\log\left(\frac{1}{\epsilon}\right)}\,d\epsilon\right). \end{aligned} \tag{20}$$

Further applying the above to Eq. (16), we get

$$G(\{\mathbf{z}_{g,i}\}) \le \frac{16\sqrt{2}\sqrt{v_\Theta \log(C_\Theta C)}}{\sqrt{N_\omega}}\left(1 + \int_0^1 \sqrt{\log\left(\frac{1}{\epsilon}\right)}\,d\epsilon\right) + 2\sqrt{2}\sqrt{\frac{\log(1/\delta)}{N_w}}.$$

Finally, from Eq. (13), we can conclude that:

$$L_0(\hat{\theta}) - L_0(\bar{\theta}^w) \le C_\rho\left(\frac{16\sqrt{2}\sqrt{v_\Theta \log(C_\Theta C)}}{\sqrt{N_\omega}}\left(1 + \int_0^1 \sqrt{\log\left(\frac{1}{\epsilon}\right)}\,d\epsilon\right) + 2\sqrt{2}\sqrt{\frac{\log(1/\delta)}{N_w}}\right)^{\frac{1}{\rho}} + \mathrm{dist}\left(\sum_{g=1}^m \omega_g P_g, P_0\right).$$

The proof of Theorem C.3 is thus finished. □