MODELDIFF: A Framework for Comparing Learning Algorithms

Harshay Shah [*1] **Sung Min Park** [*1] **Andrew Ilyas** [*1] **Aleksander Mądry** [1]

Abstract

We study the problem of *(learning) algorithm comparison*, where the goal is to find differences between models trained with two different learning algorithms. We begin by formalizing this goal as one of finding *distinguishing feature transformations*, i.e., input transformations that change the predictions of models trained with one learning algorithm but not the other. We then present MODELDIFF, a framework that leverages datamodels (Ilyas et al., 2022) to compare learning algorithms based on how they use training data. We demonstrate MODELDIFF through three case studies, comparing models trained with/without data augmentation, with/without pre-training, and with different SGD hyperparameters. Our code is available at https://github.com/MadryLab/modeldiff.

1. Introduction

Building a machine learning model involves making a number of design choices. Indeed, even after choosing a dataset, one must decide on a model architecture, an optimization method, and a data augmentation pipeline. These design choices together that define the *learning algorithm*, the function mapping training datasets to machine learning models.

One of the key reasons why these design choices matter to us is that—even when they do not directly affect accuracy—they determine the *biases* of the resulting models. For example, Hermann et al. (2020) find significant variation in shape bias (Geirhos et al., 2019) across a group of ImageNet models that vary in accuracy by less than 1%. Similarly, Liu et al. (2022) find that language models with the same loss can have drastically different implicit biases and as a result, different performances when used for transfer learning.

Motivated by this, we develop a framework for comparing learning algorithms. Our framework is *general*, as one can use it to study *any* two algorithms *without* a prior hypothesis on what the difference between them is. As a result, it enables us to precisely dissect the impact of particular design choices on model predictions. Specifically, our two main contributions are:

(a) **A precise, quantitative definition of the algorithm comparison problem.** We introduce the problem of *(feature-based) comparisons between learning algorithms*. Given two learning algorithms, the (intuitive) goal here is to find features used by one learning algorithm but not by the other. We formalize this goal by stating it as one of finding transformations in input space that induce different behavior from (i.e., that *distinguish*) models trained by the two learning algorithms (Section 2.1).

(b) **A method for comparing any two algorithms.** We propose a method, MODELDIFF, for comparing algorithms in terms of *how they use the training data*. The key tool we leverage here is *datamodeling* (Ilyas et al., 2022), which allows us to quantify the impact of each training example on a specific (single) model prediction (see Section 2.2). Intuitively, two models that use different features to arrive at their final predictions should differ in what training examples they rely on[1]. We translate this intuition into a method with formal guarantees in Section 3. The output of our method is a set of *distinguishing subpopulations*, groups of test examples for which the two algorithms systematically differ in *how* they make predictions.

We tie our method back to our formal definition of algorithm comparison in two steps. As shown in Figure 1, we first infer a pattern surfaced in each distinguishing subpopulation and use it to design an input transformation. Then, we verify the effect of this input transformation on models trained with different algorithms through counterfactual analysis.

We illustrate the utility of our framework MODELDIFF through three case studies (Section 4), motivated by typical train-time design choices one needs to make within ML pipelines.

[*]Equal contribution [1]Massachusetts Institute of Technology. Correspondence to: Harshay Shah <harshay@mit.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]For example, suppose we were comparing a texture-biased model to a shape-biased one: when faced with a test example containing, e.g., a cat in a unique pose, the texture-biased model will rely on examples with similar texture, whereas the shape-biased model will rely more on examples in a similar pose.

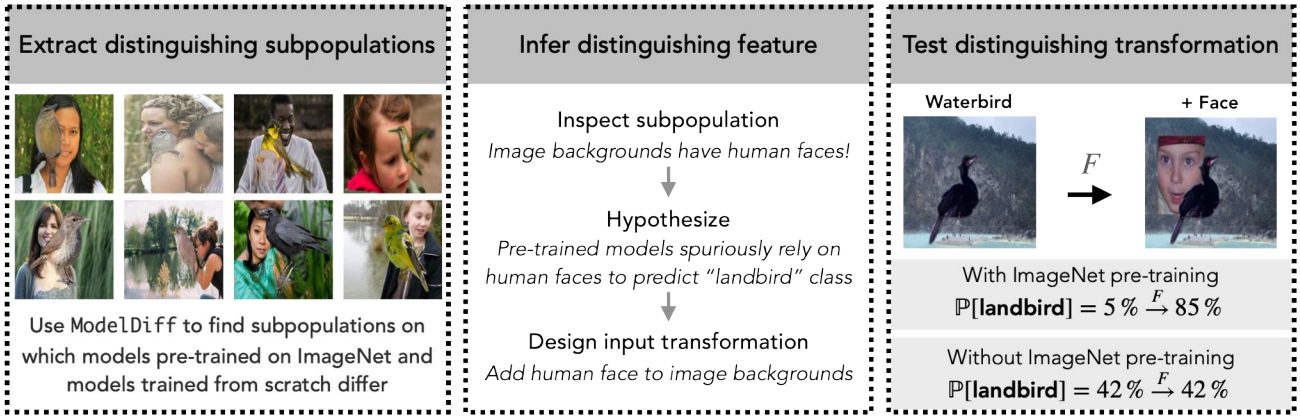


Figure 1: A demonstration of our framework. Our method (Section 3) automatically extracts coherent subpopulations of images (left) on which models differ in terms of what features they use. Then, an interpreter—whether human or automated—uses these subpopulations to infer a *feature transformation* candidate to distinguish model behavior (middle). Finally, we test the validity of this distinguishing transformation via counterfactual analysis (right).

- **Data augmentation:** We compare classifiers trained with and without data augmentation on the LIVING17 dataset (Santurkar et al., 2021). We show that models trained with data augmentation, while having higher overall accuracy, are more prone to picking up *specific* instances of co-occurrence bias and texture bias compared to models trained without data augmentation. For instance, adding a spider web to LIVING17 images increases the "spider" confidence of models trained with augmentation by 15% on average, but increases the confidence of non-augmented models by less than 1%.
- **ImageNet pre-training:** We compare ImageNet pre-trained (Deng et al., 2009; Russakovsky et al., 2015) classifiers fine-tuned on WATERBIRDS task (Sagawa et al., 2020) to classifiers trained from scratch on WATERBIRDS. We show that pre-training can suppress *or* even amplify specific spurious correlations. For example, adding a yellow patch to images changes models' average confidence in the "landbird" label by +12% (-4%) when training from scratch (pre-training). Meanwhile, adding a human face to image backgrounds changes the "landbird" confidence of pre-trained (from-scratch) models by +4% (-1%).
- **SGD hyperparameters:** We compare classifiers trained on CIFAR-10 (Krizhevsky, 2009) with different learning rates and batch sizes. Our analysis pinpoints subtle differences in model behavior induced by small changes to these hyperparameters. For example, adding a small window-like pattern to images increases the "truck" confidence by 7% and 2% on average for models trained with small and large learning rate respectively. Due to space constraints, we defer this case study to Appendix C.1.

In all three case studies, our framework surfaces fine-grained differences between models trained with different learning algorithms, enabling us to better understand the role of design choices that make up a learning algorithm.

2. Preliminaries and Setup

We first lay the groundwork for our formalization of algorithm comparison and our proposed approach. In Section 2.1, we state and formalize the algorithm comparison problem as the task of finding *distinguishing transformations* that—when applied to examples—consistently change the predictions of one model class but not the other. Then, we briefly describe *datamodels* (Ilyas et al., 2022), the key primitive behind our approach to algorithm comparison.

2.1. Formalizing algorithm comparisons

The goal of the algorithm comparison problem is to understand ways in which two learning algorithms differ in the models they yield. Specifically, we are interested in comparing the *model classes* induced by the two algorithms:

Definition 2.1 (Induced model class)**.** Given an input space $\mathcal{X}$, a label space $\mathcal{Y}$, and a model space $\mathcal{M} \subset \mathcal{X} \to \mathcal{Y}$, a learning algorithm $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^* \to \mathcal{M}$ is a (potentially random) function mapping a set of input-label pairs to a model. Fixing a data distribution $\mathcal{D}$, the model class induced by algorithm $\mathcal{A}$ is the distribution over $\mathcal{M}$ that results from applying $\mathcal{A}$ to randomly sampled datasets from $\mathcal{D}$.

Focusing on model classes (rather than individual models) isolates differences induced by the learning algorithms themselves from any inherent non-determinism in training (Zhong et al., 2021; D'Amour et al., 2020; Jiang et al., 2021). The perspective we adopt here is that model classes differ insofar as they use different features to make predictions. Thus, our goal when comparing two algorithms should be to pinpoint features that one model class uses but the other does not. Rather than try to precisely define "feature," however, we make this notion precise via functions that we call *distinguishing (feature) transformations*:

Definition 2.2 (Distinguishing transformation)**.** Let $\mathcal{A}_1, \mathcal{A}_2$ denote learning algorithms, S a dataset of input-label pairs, and $\mathcal{L}$ a loss function (e.g., correct-class margin). Suppose M_1 and M_2 are models trained on dataset $\mathcal{D}$ using algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Then, a (ϵ, δ)-distinguishing transformation of M_1 with respect to M_2 is a function $F : \mathcal{X} \to \mathcal{X}$ such that for some label $y_c \in \mathcal{Y}$,

$$\overbrace{\mathbb{E}[L_1(F(x), y_c) - L_1(x, y_c)]}^{\text{Counterfactual effect of } F \text{ on } M_1} \geq \delta \quad \text{and}$$
$$\overbrace{\mathbb{E}[L_2(F(x), y_c) - L_2(x, y_c)]}^{\text{Counterfactual effect of } F \text{ on } M_2} \leq \epsilon,$$

where $L_i(x, y) = \mathcal{L}(M_i(x), y)$ is the loss of a model trained with algorithm $\mathcal{A}_i$ on the pair (x, y), and the expectations are taken over inputs and randomness in the algorithm.

Intuitively, a distinguishing transformation, when applied to test examples, significantly changes the predictions of one model class, but not the other. Definition 2.2 also immediately suggests a way to *evaluate* the effectiveness of a "candidate" distinguishing transformation F. That is, given a hypothesis about how two algorithms differ (e.g., differences in texture bias), one can design a corresponding transformation F (e.g., style transfer in Geirhos et al. (2019)), and compare its relative effect across model classes.

***Informative* distinguishing transformations (IDTs).** Not every distinguishing transformation sheds the same amount of light on algorithm comparisons. Given two different learning algorithms, one could craft a transformation that imperceptibly modifies inputs to satisfy Definition 2.2 via adversarial perturbations or pathological out-of-distribution modifications. Such transformations satisfy Definition 2.2, but yield no benefit in qualitatively understanding the differences in *salient* features used by the model classes. In contrast, an *informative* distinguishing transformation (IDT) should (a) capture a feature that naturally arises in the data distribution and (b) be semantically meaningful.

2.2. Datamodel representations for comparison

Now, we briefly outline the datamodeling framework (Ilyas et al., 2022), and its applicability to algorithm comparison.

A primer on datamodels. Let us fix a learning algorithm $\mathcal{A}$ (i.e., a map from training datasets to machine learning models), a training set S, and a *specific* test example $x \in T$. For any subset $S' \subset S$ of the training set, we follow Ilyas et al. (2022) and define the *model output function* $f(x, S')$ as follows:

$$f(x, S') = \text{ output on } x \text{ after training a model on } S' \quad (1)$$

For example, $f(x, S')$ might represent training a deep neural network on the subset S', then computing the network's *correct-class margin* on the input x.

Ilyas et al. (2022) show that one can often approximate the (instance-wise) model output function above with simpler surrogate models called *datamodels*. For example, if the learning algorithm $\mathcal{A}$ is training a neural network on a standard image classification task and the output function f is the *correct-class margin* of the resulting classifier, a simple *linear* predictor suffices, i.e.,

$$\mathbb{E}[f(x, S')] \approx \theta_x \cdot \mathbf{1}_{S'}, \quad (2)$$

where θ_x is a parameter vector and $\mathbf{1}_{S'} \in \{0, 1\}^{|S|}$ is a binary *indicator vector* of the set S', encoding whether each example in S is included in S', i.e., $(\mathbf{1}_{S'})_i = \mathbb{1}\{x_i \in S'\}$.

The vector θ_x is the *datamodel* for the example x, and one can compute it by sampling a collection of random subsets $S_k \subset S$, training a model f on each subset S_k, and solving a regularized linear regression problem of predicting $f(x, S_k)$ from the indicator vector $\mathbf{1}_{S_k}$ for subset S_k:

$$\theta_x = \arg\min_{\theta} \sum_{k=1}^{n} \left(f(x, S_k) - \theta^\top \mathbf{1}_{S_k}\right)^2 + \lambda_x \|\theta\|_1,$$

where cross-validation determines the regularization strength $\lambda_x \in \mathbb{R}$. The resulting datamodel vectors for each test example form the *datamodel matrix* $\Theta \in \mathbb{R}^{|T| \times |S|}$.

Datamodel representations. We view the datamodel θ_j for example x_j as an $|S|$-dimensional *representation*; two properties make them useful for algorithm comparison:

(a) **Consistent basis:** The i-th coordinate of a datamodel always corresponds to the importance of the i-th training example[2], making datamodels comparable across learning algorithms (as long as the training set is fixed). In other words, datamodel representations are *inherently aligned* across different algorithms, even when models lack internal representations (e.g., decision trees).

(b) **Predictiveness:** Datamodels have a *causal* interpretation, as they can predict the counterfactual effect of removing or adding different training examples on model output for a given test example (Ilyas et al., 2022). As a result, any trends we find across the datamodel representations come with a quantitative interpretation in terms of model outputs—we return to this point in Section 3.

3. Comparing algorithms with MODELDIFF

Recall that we want to identify informative distinguishing transformations (IDTs), functions in input space that (a) induce different behavior from the two learning algorithms, (b) capture a feature arising naturally in the data, and (c) are

[2] An intuitive (but not quite accurate) way to interpret the datamodel weight θ_{ji} is as the influence (Koh & Liang, 2017; Hampel et al., 2011) of the i-th training example on test example x_j.

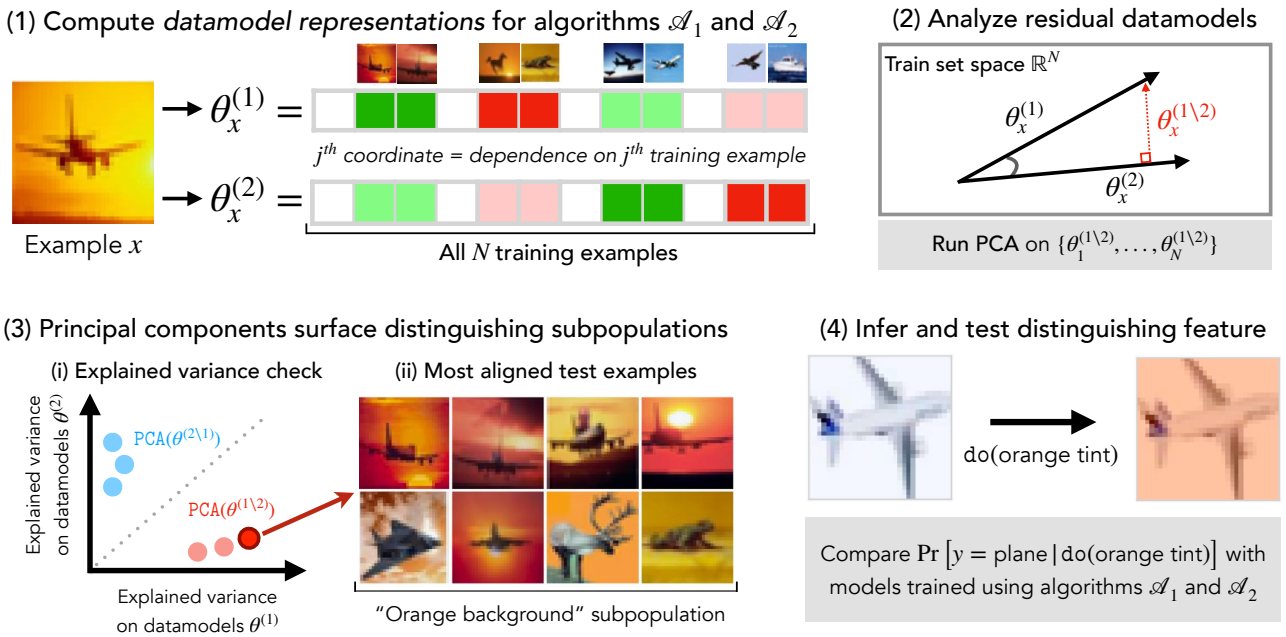


Figure 2: A visual summary of our approach. We first compute datamodels (Ilyas et al., 2022) for each algorithm. Then, we compute residual datamodels (Equation (3)) to contrast both algorithms at the example level; running PCA on these residual datamodels yields "distinguishing subpopulations." on which algorithms exhibit different behavior. Finally, we turn this subpopulation into a testable distinguishing transformation that we counterfactually verify with Definition 2.2.

semantically meaningful. Now, both the size of the transformation space and the subjective nature of "semantically meaningful" make it futile to search for IDTs directly.

Instead, our method MODELDIFF outputs a set of *distinguishing subpopulations*, groups of examples on which models trained with different algorithms exhibit different behavior. One can then use these subpopulations to infer (and test) distinguishing transformations (IDTs). MODELDIFF finds these subpopulations by examining *how each learning algorithm uses individual training examples* in a few steps:

Step 1: Compute datamodels for each algorithm. We start by computing a datamodel representation (Ilyas et al., 2022) for each example in the test set T, and then divide the datamodel by its ℓ_2 norm. The resulting normalized datamodel θ_i can be interpreted as a *direction* in train set space $\mathbb{R}^{|S|}$ that encodes how individual training examples influence the prediction on example x_i. In this step, we compute two sets of datamodels—$\Theta^{(1)}, \Theta^{(2)} \in \mathbb{R}^{|T|\times|S|}$—corresponding to learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$.

Step 2: Compute residual datamodels. For each test example x_i, we compute a *residual datamodel*, the projection of datamodel $\theta_i^{(1)}$ onto the null space of datamodel $\theta_i^{(2)}$:

$$\theta_i^{(1\setminus 2)} = \theta_i^{(1)} - \text{proj}_{\theta_i^{(2)}}\left(\theta_i^{(1)}\right) \tag{3}$$

As shown in Figure 2, the residual datamodels $\theta_i^{(1\setminus 2)}$ that contrast algorithm $\mathcal{A}_1$ from $\mathcal{A}_2$ correspond to training directions that influence $\mathcal{A}_1$ after "projecting away" the component that also influences $\mathcal{A}_2$.

Step 3(i): Run PCA on residual datamodels. Finally, we use principal component analysis to find the highest-variance directions in the space of residual datamodels:

$$\ell\text{-PCA}(\{\theta_1^{(1\setminus 2)}, \ldots, \theta_{|T|}^{(1\setminus 2)}\}) \tag{4}$$

Intuitively, the top ℓ principal components $\{u_1, \ldots u_\ell\}$ of the residual datamodels—which we call *distinguishing training directions*—correspond to (weighted) combinations of training examples that significantly influence models trained with one algorithm but not the other.

How can we verify whether these *distinguishing training directions* actually distinguish the two algorithms? Proposition 3.1 below leverages the predictiveness of datamodels (see Section 2.2) to establish this connection. Specifically, for a training direction to truly distinguish $\mathcal{A}_1$ from $\mathcal{A}_2$ (in terms of model behavior), it suffices to explain a high (resp., low) amount of the variance in datamodel representations of algorithm $\mathcal{A}_1$ (resp., $\mathcal{A}_2$).

Proposition 3.1 (Informal—see Appendix A for a formal statement and proof)**.** *Assume for illustrative purposes that Eq. (2) holds with equality (i.e., datamodels perfectly approximate model output). Let $\boldsymbol{u} \in \mathbb{R}^{|S|}$ be a principal component identified in (4), and define explained variance gap as:* $\Delta(\boldsymbol{u}) = \|\Theta^{(1)}\boldsymbol{u}\|^2 - \|\Theta^{(2)}\boldsymbol{u}\|^2$.

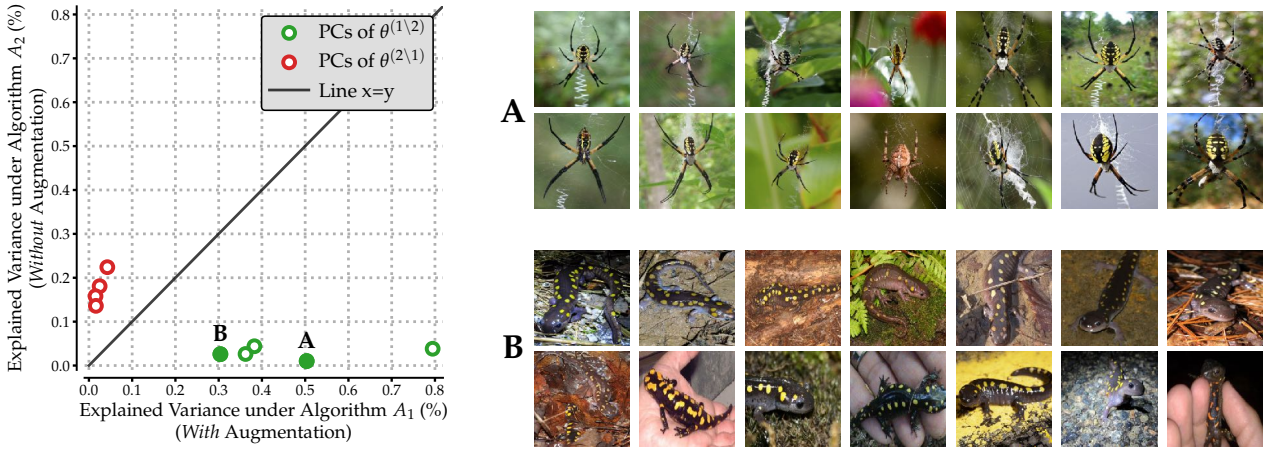


Figure 3: Comparing LIVING17 models trained with and without standard data augmentation. (Left) Each green (resp., red) point is a *training direction* (i.e., a vector $v \in \mathbb{R}^{|S|}$ representing a weighted combination of training examples) that distinguishes $\mathcal{A}_1$ from $\mathcal{A}_2$ (resp., $\mathcal{A}_2$ from $\mathcal{A}_1$) as identified by MODELDIFF (i.e., each point is a solution to (4)). The x and y coordinates of each point represent the "importance" (as given by Proposition 3.1) of the training direction to models trained with $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. **(Right)** The distinguishing subpopulations, corresponding to the distinguishing training directions annotated **A** and **B**, highlight spiders on spider webs and salamanders with yellow polka dots respectively.

Then, up/down-weighting the training set S (in a specific way) according to u will change outputs of algorithm $\mathcal{A}_1$ by $\Delta(\boldsymbol{u})$ more than the outputs of algorithm $\mathcal{A}_2$ on average (i.e., across test examples).

Importantly, these conditions can be directly verified by simply plotting the explained variances of distinguishing training direction on datamodels corresponding to each algorithm—see subplot 3(i) in Figure 2.

Step 3(ii): From distinguishing directions to distinguishing subpopulations. We now have a set of ℓ distinguishing directions u_i (i.e., principal components of the residual datamodels) that we verify by measuring $\Delta(u)$ from Proposition 3.1. The final step is to translate each direction u_i into a corresponding *distinguishing subpopulation*. Specifically, we define the i-th distinguishing subpopulation to be the set of test examples x_j whose residual datamodels $\theta_j^{(1\setminus 2)}$ most closely align with u_i—see subplot 3(ii) in Figure 2.

Connecting back to our algorithm comparison objective. MODELDIFF outputs a set of distinguishing subpopulations. We connect these back to our formal definition of algorithm comparisons (Definition 2.2) by inferring (and verifying) a distinguishing transformation from each subpopulation. [3]. For example, given the "orange background" distinguishing subpopulation in Figure 2, we can design a transformation that overlays an orange tint on inputs and measure its average effect on models trained with algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$.

[3]We intentionally leave this step open-ended to ensure our method is applicable across domains and modalities. For example, we can streamline this step in vision tasks using text-vision models like CLIP (Radford et al., 2021) (see Appendix D.2)

Figure 2 illustrates MODELDIFF (with the verification step).

4. Case studies: Applying MODELDIFF

We demonstrate our comparison framework through three case studies that study key aspects of the standard training pipeline—namely, data augmentation (4.1), pre-training (4.2), and optimizer hyperparameters (C.1). In each case study, we follow the procedure from Section 3: (i) use MODELDIFF to extract distinguishing subpopulations; (ii) inspect these subpopulations to infer candidate distinguishing transformations; and (iii) counterfactually test the effect of these transformations on models trained with different algorithms.

4.1. Case study: Data augmentation

Data augmentation is a key component of the standard computer vision training pipeline. Still, while it often improves overall performance, the effect of data augmentation on models' learned *features* remains elusive. In this case study, we study the effect of data augmentation on classifiers trained for the ImageNet-derived LIVING17 task (Santurkar et al., 2021). Specifically, we compare two classes of ResNet-18 models trained on LIVING17 with the exact same settings (see Appendix B) modulo the use of data augmentation, i.e.,

- **Algorithm** $\mathcal{A}_1$: Training with standard augmentation, (namely, horizontal flip and resized random crop with `torchvision` default parameters). Resulting models attain 89% average test accuracy (where the average is taken over randomness in training).

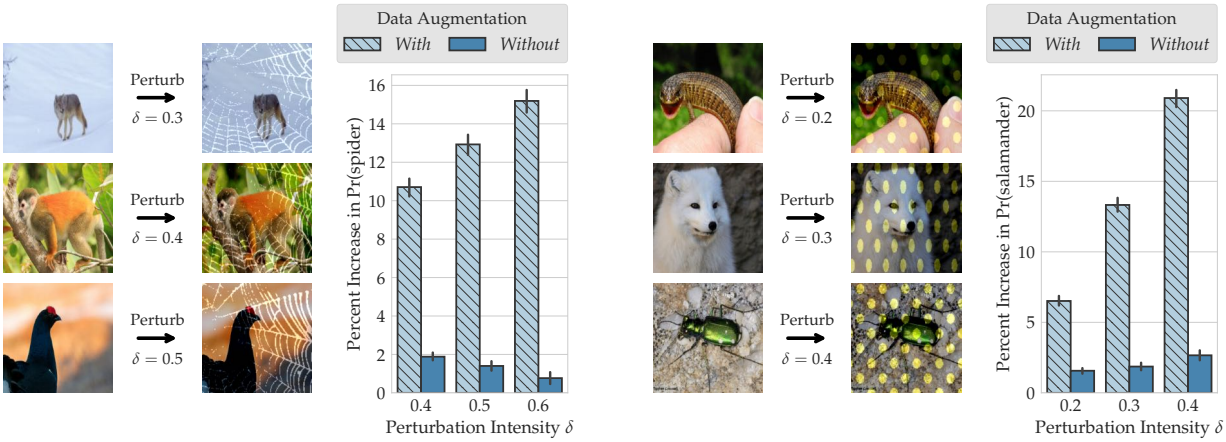


Figure 4: Effect of data augmentation on LIVING17 models. Standard data augmentation amplifies specific instances of co-occurrence (panel (a)) and texture (panel (b)) biases. The left side of each panel illustrates the distinguishing feature transformation at three different intensities δ. On the right, we plot the average effect of the transformation on the confidence of models trained with and without data augmentation, for varying δ. At moderate intensity δ, adding a spider web pattern to images makes models trained with (without) data augmentation, on average, 13% (1%) more confident in predicting the class "spider," while overlaying a polka dot pattern makes them 14% (2%) more confident in the class "salamander." In both cases, increasing the intensity δ widens the gap between the two model classes.

- **Algorithm** $\mathcal{A}_2$: Training without data augmentation. Models attain 81% average accuracy.

Applying MODELDIFF. Applying MODELDIFF to algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ gives a set of distinguishing subpopulations, visualized in Figure 3 (right). On the left side of the same figure—inspired by Proposition 3.1—we plot the variance explained by each distinguishing training direction (i.e., the solutions to (4)) in the datamodels for both $\mathcal{A}_1$ (x axis) and $\mathcal{A}_2$ (y axis). The training directions (in green) distinguishing $\mathcal{A}_1$ from $\mathcal{A}_2$ indeed explain a significant amount of variance in the datamodels of $\mathcal{A}_1$ but not in those of $\mathcal{A}_2$. Conversely, for directions distinguishing (in red) $\mathcal{A}_2$ from $\mathcal{A}_1$, the situation is reversed, as expected.

From subpopulations to IDTs. The subpopulations in Figure 3 highlight two distinguishing transformations:

- **Spider web:** Direction **A** surfaces test images that contain spiders. In contrast to random spider images in LIVING17, all images in the subpopulation contain a *spider web* in the background (see Appendix E). We thus hypothesize that models trained with standard data augmentation—moreso than those trained without it—use spider webs to predict the class "spider." We test this hypothesis with a transformation that overlays a spider web pattern onto images (see Figure 4).
- **Polka dots:** Direction **B** surfaces test images that contain salamanders. Again, comparing these images to random salamander images in LIVING17 test data reveals yellow-black polka dots as a common feature. This suggests that models trained with data augmentation rely on the polka dot texture to predict the class "salamander." To test this hypothesis, we design a transformation that adds small yellow polka dots to the images (see Figure 4).

Additional analysis in Appendix E support both hypotheses.

Findings. Figure 4 shows the effect of the above transformations on models trained with and without data augmentation. The results confirm our hypotheses. For example, overlaying a spider web pattern with 30% opacity increases P("spider")—average softmax confidence in the spider label—for models trained with (without) data augmentation by 11% (1%). Similarly, overlaying the yellow polka dot texture with 30% opacity increased P("salamander") by 14% (2%). Furthermore, increasing the transformations' opacity consistently widens the gap between the predictions of the two model classes. Overall, these differences verify that the inferred distinguishing transformations indeed differentiate the two algorithms as per Definition 2.2.

Connection to previous work. Our case study shows how training with standard data augmentation can amplify *specific* instances of co-occurrence bias (spider webs) and texture bias (polka dots). These findings empirically support the view of data augmentation as *feature manipulation* (Shen et al., 2022). Moreover, the distinguishing transformations substantiate that data augmentation can (a) introduce unwanted biases (Hermann et al., 2020)) and (b) disparately impact performance across classses (Balestriero et al., 2022).

4.2. Case study: Pre-training

Pre-training on large datasets is a standard approach to improve performance on downstream tasks with limited training data. Here, we study the effect of ImageNet pre-

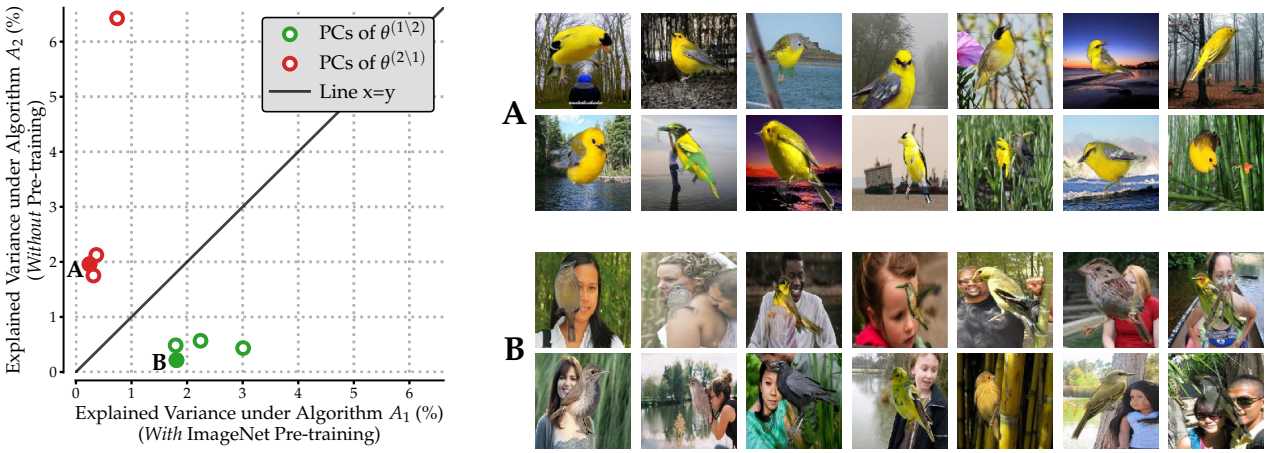


Figure 5: Comparing Waterbirds models trained with and without ImageNet pre-training. An analog to Figure 3 for our second case study, see Figure 3 for a description. In this case, subpopulation **A** surfaces images of yellow birds and subpopulation **B** surfaces images with human faces in the background. Additional subpopulations shown in Appendix G.5.

training (Deng et al., 2009) on models trained for the WATERBIRDS task of classifying images of birds as "waterbird" or "landbird" (Sagawa et al., 2020). We compare two classes of ResNet-50 models trained with the exact same settings (see Appendix B) modulo the use of ImageNet pre-training:

- **Algorithm** $\mathcal{A}_1$: ImageNet pre-trained models fine-tuned on WATERBIRDS. Models attain 89.1% accuracy.
- **Algorithm** $\mathcal{A}_2$: Training directly on WATERBIRDS. Models attain 63.9% average accuracy.

Identifying IDTs with MODELDIFF. Like in Section 4.1, we apply MODELDIFF with results shown in Figure 5; the subpopulations surface two distinguishing transformations:

- **Yellow color:** Subpopulation **A** comprises test images of yellow birds belonging to class "landbird." This leads us to hypothesize that models trained from scratch spuriously rely on the color yellow to predict the class "landbird," whereas ImageNet pre-trained models do not. To test this hypothesis, we design a transformation that adds a yellow square patch to images (see Figure 6).
- **Human face:** Subpopulation **B** comprises "landbird" images with *human faces* in the background, suggesting that ImageNet pre-training introduces a spurious dependence on faces to predict the label "landbird." We test this hypothesis with a transformation that adds human faces to the background of WATERBIRDS images (see Figure 6).

Analysis in Appendix E further supports both hypotheses.

Findings. In Figure 6, we evaluate the effect of the above distinguishing transformations on models trained with and without ImageNet pre-training. The results confirm both hypotheses. Adding a small (40px) yellow square patch to test images increased $P(\text{"landbird"})$ by 12% for models trained from scratch but *decreased* $P(\text{"landbird"})$ for models pre-trained on ImageNet. Similarly, adding a human face patch[4] to image backgrounds increased $P(\text{"landbird"})$ by up to 4% for pre-trained models, but did not significantly affect models trained from scratch. Once again, increasing the intensity (i.e., patch size) of these transformations further widens the gap in sensitivity between the model classes.

Streamlining the verification step with CLIP. So far, we have shown that manual inspection of distinguishing subpopulations suffices to infer (and verify) distinguishing transformations (IDTs). Nevertheless, depending on the dataset and modality, one can also domain-specific tools to infer IDTs from subpopulations in a streamlined manner. For example, in Appendix D.2, we use CLIP embeddings (Radford et al., 2021) in order to search over natural language descriptions (or captions) that best contrast distinguishing subpopulations from the entire dataset. In this case, for subpopulations **A** and **B**, our CLIP-based approach in Appendix D.2 generates candidate hypotheses {"yellow", "canary", "lemon"} and {"florists", "florist", "faces"} respectively.

Connections to prior work. Our results show that ImageNet pre-training reduces dependence on *specific* spurious correlations (e.g., yellow color $\rightarrow$ landbird) but also introduces new ones (e.g., human face $\rightarrow$ landbird). These findings connect to two (seemingly contradictory) phenomena where pre-training improves robustness to spurious features (Ghosal et al., 2022; Tu et al., 2020) while also transferring over spurious correlations (Salman et al., 2022; Neyshabur et al., 2020) from the pre-training dataset.

[4]We extract human face patches from ImageNet examples using face annotations from Yang et al. (2022) (see Appendix B.4).

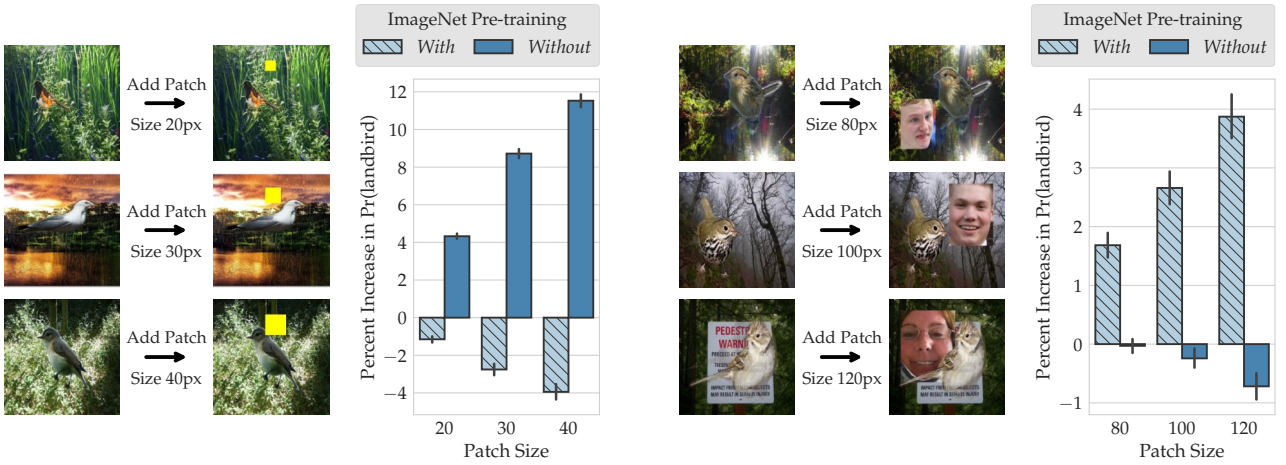


Figure 6: Effect of ImageNet pre-training on WATERBIRDS classification. Analogously to Figure 4, we use our framework to identify spurious correlations that are either suppressed or amplified by ImageNet pre-training. **(Left)** Adding a yellow patch to images makes models trained without (with) pre-training, on average, 9% more (2% less) confident in predicting the label "landbird." **(Right)** Adding human faces to image backgrounds makes models trained with (without) pre-training, on average, 3% (0%) more confident in predicting the label "landbird." Again, in both cases, increasing the intensity of feature transformations widens the gap in treatment effect between the two model classes.

5. Related work

We now compare and contrast our approach to algorithm comparison with approaches to the related problem of model comparison, where one tries to characterize the difference between two (usually fixed) machine learning models.

Representation-based comparison. A popular approach in deep learning is to compare two *fixed* models using their internal representations. Unlike datamodels, these representations lack a consistent interpretation across models (see Section 2.2). Consequently, representation-based comparisons typically quantify the degree to which different models' representations can be aligned (Raghu et al., 2017; Kornblith et al., 2019; Bansal et al., 2021; Chen et al., 2021). For example, prior works use these methods to compare architectures (Raghu et al., 2021; Nguyen et al., 2021) and language models (Wu et al., 2020). More recently, however, Ding et al. (2021) and Davari et al. (2022) show that these methods are not reliable for testing *functional* differences between models. In contrast, our work differs in terms of problem (learning algorithms) as well as objective (distinguishing transformations)—see Appendix H for details.

Comparing feature attributions. Another line of work compares models in terms of how they use features at test time. In the presence of a known set of features, one can compute feature importances (e.g., via SHAP (Lundberg & Lee, 2017)) and compare them across models (Wang et al., 2022). In the absence of known features, one can potentially use instance-level explanation methods such as saliency maps. Furthermore, common explanation methods (a) generally do not help at distinguishing models (Denain & Steinhardt, 2022) and (b) often fail at accurately highlighting features learned by the model (Adebayo et al., 2018; Hooker et al., 2018; Shah et al., 2021).

Example-level comparisons. An alternative method for comparing models is to compare their predictions directly. Zhong et al. (2021) compare predictions of small and large language models on a per-example level to find that larger models are not uniformly better across examples. Similarly, Mania et al. (2019) study the *agreement* between models, i.e., how often they output the same prediction on a per-example level. Meding et al. (2022) show that filtering out "impossible" and "trivial" test examples amplifies prediction-level variations between models. In contrast, MODELDIFF leverages datamodels to trace example-level predictions back to training data and subsequently identify IDTs.

We defer additional related work to Appendix H.

6. Discussion

We discuss some ways to extend and evaluate MODELDIFF.

Measuring overall similarity between two algorithms. In Appendix D.1, we adapt MODELDIFF to quantify the similarity between learning algorithms in aggregate. Specifically, given example x, the cosine similarity between its datamodels (one for each algorithm) measures how similarly training examples influence the two model classes' predictions. Then, aggregating over all x gives a distribution that quantifies the overall similarity of two learning algorithms.

Comparing learning algorithms at the prediction level. One might ask whether simply analyzing differences in predictions across learning algorithms suffices to surface similar distinguishing subpopulations. To investigate this, we

analyze whether filtering out examples on which predictions differ across algorithms has an impact on the distinguishing directions extracted via MODELDIFF. In Appendix G.4, we find that even after controlling for prediction-level differences, MODELDIFF outputs similar distinguishing directions (and subpopulations). This suggests that MODELDIFF finds differences finer-grained than at the prediction-level.

Evaluating MODELDIFF subpopulations. We also design a controlled setting to *directly* evaluate MODELDIFF subpopulations (i.e., without needing to infer a distinguishing transformation). Specifically, we use spurious correlations to plant a "ground-truth" distinguishing subpopulation on which two algorithms learn different features to make similar predictions. In Appendix C.2, we show that MODELDIFF *fully* recovers the planted distinguishing subpopulation.

Speeding up MODELDIFF. The computational cost of our approach scales with the number of models trained to compute datamodels. In Appendix G.2, we show that one can extract similar distinguishing subpopulations even with $10\times$ fewer models to estimate datamodels). Moreover, the "black-box" usage of datamodels implies that any improvements in the sample efficiency of datamodel estimation would directly speed up MODELDIFF comparisons.

7. Conclusion

We introduce a framework for fine-grained comparison of any two learning algorithms. We first formalize algorithm comparison as the problem of finding informative distinguishing transformations (IDTs). Then, we present MODELDIFF, which compares models trained using two different algorithms by contrasting how these models *rely on training data* to make predictions. Through three case studies, we showcase the utility of our framework in pinpointing how key aspects of ML pipelines can shape model behavior.

8. Acknowledgements

Work supported in part by the NSF grants CCF-1553428 and CNS-1815221, and Open Philanthropy. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0015. Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Abid, A., Yuksekgonul, M., and Zou, J. Meaningfully debugging model mistakes using conceptual counterfactual explanations. In *arXiv preprint arXiv:2106.12723*, 2022.

Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., and Kim, B. Sanity checks for saliency maps. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Balestriero, R., Bottou, L., and LeCun, Y. The effects of regularization and data augmentation are class dependent. *arXiv preprint arXiv:2204.03632*, 2022.

Bansal, Y., Nakkiran, P., and Barak, B. Revisiting model stitching to compare neural representations. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Baradad Jurjo, M., Wulff, J., Wang, T., Isola, P., and Torralba, A. Learning to see by looking at noise. *Advances in Neural Information Processing Systems*, 34:2556–2569, 2021.

Chen, Z., Lu, Y., Hu, J., Yang, W., Xuan, Q., Wang, Z., and Yang, Z. Revisit similarity of neural network representations from graph perspective. *arXiv preprint arXiv:2111.11165*, 2021.

Csiszarik, A., Korosi-Szabo, P., Matszangosz, A., Papp, G., and Varga, D. Similarity and matching of neural network representations. In *Neural Information Processing Systems (NeurIPS)*, 2021.

Cui, T., Kumar, Y., Marttinen, P., and Kaski, S. Deconfounded representation similarity for comparison of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Dabkowski, P. and Gal, Y. Real time image saliency for black box classifiers. In *Neural Information Processing Systems (NeurIPS)*, 2017.

D'Amour, A., Heller, K. A., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M. D., Hormozdiari, F., Houlsby, N., Hou, S., Jerfel, G., Karthikesalingam, A., Lucic, M., Ma, Y., McLean, C. Y., Mincu, D., Mitani, A., Montanari, A., Nado, Z., Natarajan, V., Nielson, C., Osborne, T. F., Raman, R., Ramasamy, K., Sayres, R., Schrouff, J., Seneviratne, M., Sequeira, S., Suresh, H., Veitch, V., Vladymyrov, M., Wang, X., Webster, K., Yadlowsky, S., Yun, T., Zhai, X., and Sculley, D. Underspecification presents challenges for credibility in modern machine learning. In *Arxiv preprint arXiv:2011.03395*, 2020.

Davari, M., Horoi, S., Natik, A., Lajoie, G., Wolf, G., and Belilovsky, E. Reliability of cka as a similarity measure in deep learning. *arXiv preprint arXiv:2210.16156*, 2022.

Denain, J.-S. and Steinhardt, J. Auditing visualizations: Transparency methods struggle to detect anomalous behavior. *arXiv preprint arXiv:2206.13498*, 2022.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition (CVPR)*, 2009.

Ding, F., Denain, J.-S., and Steinhardt, J. Grounding representation similarity with statistical testing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Eyuboglu, S., Varma, M., Saab, K., Delbrouck, J.-B., Lee-Messer, C., Dunnmon, J., Zou, J., and Ré, C. Domino: Discovering systematic errors with cross-modal embeddings. *arXiv preprint arXiv:2203.14960*, 2022.

Feldman, V. and Zhang, C. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 2881–2891, 2020.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations (ICLR)*, 2019.

Ghorbani, A., Wexler, J., Zou, J., and Kim, B. Towards automatic concept-based explanations. *arXiv preprint arXiv:1902.03129*, 2019.

Ghosal, S. S., Ming, Y., and Li, Y. Are vision transformers robust to spurious correlations? *arXiv preprint arXiv:2203.09125*, 2022.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons, 2011.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Hermann, K., Chen, T., and Kornblith, S. The origins and prevalence of texture bias in convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2020.

Hoffer, E., Hubara, I., and Soudry, D. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Hooker, S., Erhan, D., Kindermans, P.-J., and Kim, B. A benchmark for interpretability methods in deep neural networks. *arXiv preprint arXiv:1806.10758*, 2018.

Ilyas, A., Park, S. M., Engstrom, L., Leclerc, G., and Madry, A. Datamodels: Predicting predictions from training data. In *International Conference on Machine Learning (ICML)*, 2022.

Jain, S., Lawrence, H., Moitra, A., and Madry, A. Distilling model failures as directions in latent space. *arXiv preprint arXiv:2206.14754*, 2022.

Jiang, Y., Nagarajan, V., Baek, C., and Kolter, J. Z. Assessing generalization of sgd via disagreement. In *Arxiv preprint arXiv:2106.13799*, 2021.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International conference on machine learning (ICML)*, 2018.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017.

Kornblith, S., Norouzi, M., Lee, H., and Hinton, G. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

Krizhevsky, A. Learning multiple layers of features from tiny images. In *Technical report*, 2009.

Leclerc, G., Salman, H., Ilyas, A., Vemprala, S., Engstrom, L., Vineet, V., Xiao, K., Zhang, P., Santurkar, S., Yang, G., et al. 3db: A framework for debugging computer vision models. In *arXiv preprint arXiv:2106.03805*, 2021.

Leclerc, G., Ilyas, A., Engstrom, L., Park, S. M., Salman, H., and Madry, A. ffcv. `https://github.com/libffcv/ffcv/`, 2022.

Li, Y., Wei, C., and Ma, T. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Neural Information Processing Systems (NeurIPS)*. 2019.

Liu, H., Xie, S. M., Li, Z., and Ma, T. Same pre-training loss, better downstream: Implicit bias matters for language models. In *arXiv preprint arXiv:2210.14199*, 2022.

Lundberg, S. and Lee, S.-I. A unified approach to interpreting model predictions. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Mania, H., Miller, J., Schmidt, L., Hardt, M., and Recht, B. Model similarity mitigates test set overuse. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9993–10002, 2019.

Meding, K., Buschoff, L. M. S., Geirhos, R., and Wichmann, F. A. Trivial or impossible — dichotomous data difficulty masks model differences (on imagenet and beyond). In *International Conference on Learning Representations (ICLR)*, 2022.

Miller, G. A. Wordnet: a lexical database for english. *Communications of the ACM*, 1995.

Morcos, A., Raghu, M., and Bengio, S. Insights on representational similarity in neural networks with canonical correlation. *Advances in Neural Information Processing Systems*, 31, 2018a.

Morcos, A. S., Barrett, D. G., Rabinowitz, N. C., and Botvinick, M. On the importance of single directions for generalization. In *International Conference on Learning Representations (ICLR)*, 2018b.

Neyshabur, B., Sedghi, H., and Zhang, C. What is being transferred in transfer learning? *Advances in neural information processing systems*, 33:512–523, 2020.

Nguyen, T., Raghu, M., and Kornblith, S. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations (ICLR)*, 2021.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *arXiv preprint arXiv:2103.00020*, 2021.

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, 2017.

Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., and Dosovitskiy, A. Do vision transformers see like convolutional neural networks? In *Neural Information Processing Systems (NeurIPS)*, 2021.

Ribeiro, M. T., Singh, S., and Guestrin, C. " why should i trust you?" explaining the predictions of any classifier. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

Ruiz, N., Bargal, S. A., Xie, C., Saenko, K., and Sclaroff, S. Finding differences between transformers and convnets using counterfactual simulation testing. *arXiv preprint arXiv:2211.16499*, 2022.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. In *International Journal of Computer Vision (IJCV)*, 2015.

Sagawa, S., Koh, P. W., Hashimoto, T. B., and Liang, P. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *International Conference on Learning Representations*, 2020.

Salman, H., Jain, S., Ilyas, A., Engstrom, L., Wong, E., and Madry, A. When does bias transfer in transfer learning? In *arXiv preprint arXiv:2207.02842*, 2022.

Santurkar, S., Tsipras, D., and Madry, A. Breeds: Benchmarks for subpopulation shift. In *International Conference on Learning Representations (ICLR)*, 2021.

Shah, H., Jain, P., and Netrapalli, P. Do input gradients highlight discriminative features? *Advances in Neural Information Processing Systems*, 34, 2021.

Shen, R., Bubeck, S., and Gunasekar, S. Data augmentation as feature manipulation. In *International Conference on Machine Learning*, pp. 19773–19808. PMLR, 2022.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

Singla, S. and Feizi, S. Salient imagenet: How to discover spurious features in deep learning? In *International Conference on Learning Representations*, 2021.

Tu, L., Lalwani, G., Gella, S., and He, H. An empirical study on robustness to spurious correlations using pretrained language models. *Transactions of the Association for Computational Linguistics*, 8:621–633, 2020.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. The caltech-ucsd birds-200-2011 dataset. 2011.

Wang, J., Wang, L., Zheng, Y., Yeh, C.-C. M., and andWei Zhang, S. J. Learning-from-disagreement: A model comparison and visual analytics framework. *arXiv preprint arXiv:2201.07849*, 2022.

Wen, Y., Luk, K., Gazeau, M., Zhang, G., Chan, H., and Ba, J. An empirical study of large-batch stochastic gradient descent with structured covariance noise. *arXiv preprint arXiv:1902.08234*, 2019.

Wong, E., Santurkar, S., and Madry, A. Leveraging sparse linear layers for debuggable deep networks. In *International Conference on Machine Learning (ICML)*, 2021.

Wu, J., Belinkov, Y., Sajjad, H., Durrani, N., Dalvi, F., and Glass, J. Similarity analysis of contextual word representation models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

Xiao, K., Engstrom, L., Ilyas, A., and Madry, A. Noise or signal: The role of image backgrounds in object recognition. *arXiv preprint arXiv:2006.09994*, 2020.

Yang, K., Yau, J. H., Fei-Fei, L., Deng, J., and Russakovsky, O. A study of face obfuscation in imagenet. In *International Conference on Machine Learning*, pp. 25313–25330. PMLR, 2022.

Zhong, R., Ghosh, D., Klein, D., and Steinhardt, J. Are larger pretrained language models uniformly better? Comparing performance at the instance level. In *Findings of the Association for Computational Linguistics (Findings of ACL)*, 2021.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. Places: A 10 million image database for scene recognition. In *IEEE transactions on pattern analysis and machine intelligence*, 2017.

Appendices

A Algorithm analysis **14**

B Experiment Setup **15**

B.1 Datasets . . . 15

B.2 Models, learning algorithms, and hyperparameters . . . 15

B.3 Datamodels . . . 16

B.4 Distinguishing feature transformations . . . 17

B.5 Training infrastructure . . . 17

C Additional case studies **18**

C.1 SGD hyperparameters . . . 18

C.2 Synthetic spurious correlations . . . 20

D Extending MODELDIFF **22**

D.1 Aggregate metric for algorithm comparison . . . 22

D.2 Leveraging CLIP to analyze distinguishing subpopulations . . . 23

E Additional analysis of distinguishing subpopulations **25**

E.1 Case study: Standard data augmentation . . . 26

E.2 Case study: ImageNet pre-training . . . 27

E.3 Case study: SGD hyperparameters . . . 28

F Additional evaluation of distinguishing transformations **30**

F.1 Case study: Standard data augmentation . . . 30

F.2 Case study: ImageNet pre-training . . . 31

F.3 Case study: SGD hyperparameters . . . 32

G Miscellaneous results **33**

G.1 Explained variance of residual datamodel principal components . . . 33

G.2 Effect of sample size on datamodel estimation . . . 34

G.3 Algorithm comparisons with penultimate-layer representations . . . 36

G.4 Effect of prediction-level differences on distinguishing training directions . . . 38

G.5 Top-k subpopulations surfaced by principal components of residual datamodels . . . 39

H Additional related work **40**

A. Algorithm analysis

Here, we give a more formal version of Proposition 3.1. First, we need to define the sense in which a given learning algorithm is sensitive to up/down-weighting the training set according to a training direction $u \in \mathbb{R}^{|S|}$.

Sensitivity along a training direction. We consider the following probabilistic notion of sensitivity: we measure how $f(x, S')$ varies in expectation, if relative to some base distribution $\mathcal{D}_0$ over S', we up/down weight the probability of each training example by its weight u_i.

More specifically, for $\mathcal{D}_0$, we consider sampling S' uniformly, i.e., by choosing each element of the full training set S with probability 1/2. Then, we consider the perturbed distribution $\mathcal{D}_u$ where x_i is sampled with probability $(1 + u_i)/2$. Finally, we define the *sensitivity* of $f(x, \cdot)$ along u (or *u-sensitivity*) as:

$$f(x, \cdot)|_u = (\mathbb{E}_{S' \sim \mathcal{D}_u} f(x, S') - \mathbb{E}_{S' \sim \mathcal{D}_0} f(x, S'))^2$$

where we take the square as we are interested in the magnitude.

We now give the formal result connecting the explained variance gap in datamodels (which we empirically observed in our case studies in Section 4) to the above definition of senstivity.

Theorem A.1 (Formal version of Proposition 3.1)**.** *Consider two learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ applied to a training set S and evaluated on a test set T. Assume that their datamodels $\theta_x^{(1)}$ and $\theta_x^{(2)}$ perfectly approximate the respective model outputs $f^{(1)}(x, \cdot)$ and $f^{(2)}(x, \cdot)$. Then, for a given training direction $u \in \mathbb{R}^{|S|}$, the explained variance gap*

$$\Delta(\boldsymbol{u}) = \|\Theta^{(1)}\boldsymbol{u}\|^2 - \|\Theta^{(2)}\boldsymbol{u}\|^2$$

is equal to the following quantity:

$$\sum_{x \in T} f^{(1)}(x, \cdot)|_u - \sum_{x \in T} f^{(2)}(x, \cdot)|_u$$

which is the difference in u-sensitivity of model outputs of algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$.

Proof. With our linear assumption and definition of sensitivity, the proof is almost immediate. First, under the linearity assumption, the sensitivity $f(x_i, \cdot)|_u$ just reduces to $(\theta_i \cdot u)^2$. It follows that the total sensitivity, $\sum_i f(x_i, \cdot)|_u$ is just $\sum_i (\theta_i \cdot u)^2$, from which the claim follows.

□

B. Experiment Setup

In this section, we outline the experimental setup—datasets, models, training algorithms, hyperparameters, and datmodels—used for our case studies in Section 4.

B.1. Datasets

Living17. The Living17 dataset (Santurkar et al., 2021) is an ImageNet-derived dataset, where the task is to classify images belonging to 17 types of living organisms (e.g., salamander, bear, fox). Each Living17 class corresponds to an ImageNet superclass (i.e., a set of ImageNet classes aggregated using WordNet (Miller, 1995)). Santurkar et al. (2021) introduce Living17 as one of four benchmark to evaluate model robustness to realistic subpopulation shifts. In our case study, we study the effect of data augmentation using a variant of this dataset, wherein the training and test images belong to the same set of subpopulations (i.e., no subpopulation shift).

Waterbirds. The Waterbirds dataset (Sagawa et al., 2020) consists of bird images taken from the CUB dataset (Wah et al., 2011) and pasted on backgrounds from the Places dataset (Zhou et al., 2017). The task here is to classify "waterbirds" and "landbirds" in the presence of spurious correlated "land" and "water" backgrounds in the training data. Sagawa et al. (2020) introduce Waterbirds as a benchmark to evaluate models under subpopulation shifts induced by spurious correlations. In our case study, we compare how models trained from scratch on Waterbirds data differ from ImageNet-pretrained models that are fine-tuned on Waterbirds data.

CIFAR-10. We consider the standard CIFAR-10 (Krizhevsky, 2009) image classification dataset in order to study the effect of two SGD hyperparameters: learning rate and batch size.

Summary statistics of the datasets described above are outlined in Table 1.

Table 1: Summary statistics of datasets

Dataset	Classes	Size (Train/Test)	Input Dimensions
Living17	17	88,400/3,400	$3 \times 224 \times 224$
Waterbirds	2	4,795/5,794	$3 \times 224 \times 224$
CIFAR-10	10	50,000/10,000	$3 \times 32 \times 32$

B.2. Models, learning algorithms, and hyperparameters

Living17. We use the standard ResNet18 architecture (He et al., 2015) from the `torchvision` library. We train models for 25 epochs using SGD with the following configuration: initial learning rate 0.6, batch size 1024, cyclic learning rate schedule (with peak at epoch 12), momentum 0.9, weight decay 0.0005, and label smoothing (with smoothing hyperparameter 0.1). To study the effect of data augmentation, we train models with the following algorithms:

- **Algorithm $\mathcal{A}_1$ (with data augmentation)**: Models are trained with standard data augmentation: random resized cropping (with default `torchvision` hyperparamters) and random horizontal flips. On average, models attain 89.2% average test accuracy.
- **Algorithm $\mathcal{A}_2$ (without data augmentation)**: Models are trained without data augmentation. On average, models attain 81.9% average test accuracy.

Waterbirds. We use the standard ResNet50 architecture from the `torchvision` library. We train models using SGD with momentum 0.9 and weight decay 0.0001 for a maximum of 50 epochs (and stop early if the training loss drops below 0.01). For model selection, we choose the model checkpoint that has the maximum average accuracy on the validation dataset. As in Sagawa et al. (2020), we do not use data augmentation. In our case study on pre-training, we consider ImageNet pre-trained models from `torchvision`. We consider models trained using the following algorithms:

- **Algorithm $\mathcal{A}_1$ (ImageNet pre-training)**: Models pre-trained on ImageNet are fully fine-tuned on Waterbirds data with a fixed SGD learning rate 0.005 and batch size 64. On average, models attain 89.1% (non-adjusted) average test accuracy and 63.9% worst-group test accuracy.

- **Algorithm $\mathcal{A}_2$ (Training from scratch)**: Models are trained from scratch (i.e., random initialization) on Waterbirds data with SGD: initial learning rate 0.01, batch size 64, and a linear learning rate schedule ($0.2\times$ every 15 epochs). On average, models attain 63.6% average test accuracy and 5.7% worst-group test accuracy.

CIFAR-10. We use the ResNet9 architecture from Kakao Brain[5], which is optimized for fast training. We train models using SGD with momentum 0.9 and weight decay 0.0005 for a maximum of 100 epochs (and stop early if the training loss drops below 0.01). We augment training data with a standard data augmentation scheme: random resized cropping with 4px padding and random horizontal flips. To study the effect of SGD noise in our case study, we vary learning rate and batch size. Specifically, we compare models trained with the following algorithms:

- **Algorithm $\mathcal{A}_1$ (high SGD noise)**: Models are trained with SGD using a large initial learning rate (0.1), small batch size (256), and a linear learning rate schedule ($0.5\times$ every 20 epochs). On average, models attain 93.3% test accuracy.
- **Algorithm $\mathcal{A}_2$ (low SGD noise)**: Models are trained with SGD using a small fixed learning rate (0.02) and large batch size (1024). On average, models attain 89.5% test accuracy.

B.3. Datamodels

Now, we provide additional details on datamodels which, we recall, are computed in the first step of our algorithm comparison framework.

Estimating linear datamodels. Recall that the datamodel vector for example x_j, $\theta_j^{(i)} \in \mathbb{R}^{|S|}$, encodes the importance of individual training examples S to model's loss at example x_j when trained with algorithm $\mathcal{A}_i$. Concretely, given test example x_j and training set $S = \{x_1, \ldots, x_d\}$, the datamodel θ_j is a sparse linear model (or surrogate function) trained on the following regression task: For a training subset $S' \subset S$, can we predict the correct-class margin $f_{\mathcal{A}}(x_j; S')$ of a model trained on S' with algorithm $\mathcal{A}$? This task can be naturally formulated as the following supervised learning problem: Given a training set $\{(S_i, f_{\mathcal{A}}(x; S_i))\}_{i=1}^m$ of size m, the datamodel θ_j (for example x_j) is the solution to the following problem:

$$\theta_j = \min_{w \in \mathbb{R}^{|S|}} \frac{1}{m} \sum_{i=1}^{m} \left(w^\top \mathbf{1}_{S_i} - f_{\mathcal{A}}(x_j; S_i) \right)^2 + \lambda \|w\|_1, \tag{5}$$

where $\mathbf{1}_{S_i}$ is a boolean vector that indicates whether examples in the training dataset $x \in S$ belong to the training subset S_i. Note that each datamodel training point $(S_i, f_{\mathcal{A}}(x_j, S_i))$ is obtained by (a) training a model f (e.g., ResNet9) on a subset of data S_i (e.g., randomly subsampled CIFAR data) and (b) evaluating the trained model's output on example x_j.

Datamodel estimation hyperparameters. Recall that our algorithm comparison framework in Section 3 involves estimating two sets of datamodels $\{\theta^{(1)}\}$ and $\{\theta^{(2)}\}$ for learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. In our case studies, we estimate two datamodels, $\theta_i^{(1)}$ and $\theta_i^{(2)}$ for every example x_i in the test dataset. Estimating these datamodels entail three design choices:

- **Sampling scheme for train subsets:** Like in Ilyas et al. (2022), we use α-random subsets of the training data, where α denotes the subsampling fraction; we set $\alpha = 50\%$ as it maximizes sample efficiency (or model reuse) for empirical influence estimation (Feldman & Zhang, 2020), which is equivalent to a variant of linear datamodels (Ilyas et al., 2022).
- **Sample size for datamodel estimation:** Recall that a datamodel training set of size m corresponds to training m models (e.g., m ResNet18 models on CIFAR-10) on independently sampled train subsets (or masks). We estimate datamodels on LIVING17, WATERBIRDS, and CIFAR-10 using $120k$, $50k$, and $50k$ samples (or models) per learning algorithm respectively; we make a $90 - 10\%$ train-validation split.
- ℓ_1 **sparsity regularization:** We use cross-validation to select the sparsity regularization parameter λ. Specifically, for each datamodel, we evaluate the MSE on a validation split to search over $k = 50$ logarithmically spaced values for λ along the regularization path. As in (Ilyas et al., 2022), we then re-compute the datamodel on the entire dataset with the optimal λ value and all m training examples.

[5] https://github.com/wbaek/torchskeleton/blob/master/bin/dawnbench/cifar10.py

B.4. Distinguishing feature transformations

We counterfactually verify distinguishing feature transformations (inferred from distinguishing subpopulations) by evaluating whether the corresponding transformations change model behavior as hypothesized. Here, we describe the feature transformations used in Section 4 in more detail[6].

Designing feature transformations. We design feature transformations that modify examples by adding a specific patch or perturbation. We vary the intensity of patch-based and perturbation-based transformations via patch size k and perturbation intensity δ respectively. Additional details specific to each case study:

- **Pre-training.** We use patch-based transformations in this case. For the yellow color feature, we add a $k \times k$ square yellow patch to the input. For the human face feature, we add a $k \times k$ image of a human face to the input. To avoid occlusion with objects in the image foreground, we add the human face patch to the background. We make a bank of roughly 300 human faces using ImageNet face annotations (Yang et al., 2022) by (a) cropping out human faces from ImageNet validation examples and (b) manually removing mislabeled, low-resolution, and unclear human face images.
- **Data augmentation**. We design perturbation-based transformations to verify the identified distinguishing features: spider web and polka dots. In both cases, we δ-perturb each input with a random crop of a fixed grayscale spider web or yellow polka dot pattern.
- **SGD hyperparameters.** We use patch-based transformations in this case study. For the black-white texture feature, we add a k-sized patch that loosely resembles a black-white dog nose. Similarly, for the rectangular shape feature, we add a k-sized patch that loosely resembles windows.

Evaluating feature transformations. As shown in Section 4, given two learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$, we evaluate whether a feature transformation F changes predictions of models trained with $\mathcal{A}_1$ and $\mathcal{A}_2$ as hypothesized. To evaluate the counterfactual effect of transformation F on model M, we evaluate the extent to which applying F to input examples x increases the confidence of models in a particular class y. In our experiments, we estimate this counterfactual effect by averaging over all test examples and over 500 models trained with each learning algorithm.

B.5. Training infrastructure

Data loading. We use `FFCV`[7] (Leclerc et al., 2022), which removes the data loading bottleneck for smaller models, gives a 3-4$\times$ improvement in throughput (i.e., number of models a day per GPU).

Datamodels regression. In addition to `FFCV`, we use the `fast-l1` package—a SAGA-based GPU solver for ℓ_1-regularized regression—to parallelize datamodel estimation.

Computing resources. We train our models on a cluster of machines, each with 9 NVIDIA A100 or V100 GPUs and 96 CPU cores. We also use half-precision to increase training speed.

[6]The code for these feature transformations is available at `https://github.com/MadryLab/modeldiff`.
[7]Webpage: `http://ffcv.io`

C. Additional case studies

C.1. SGD hyperparameters

The choices of optimizer and corresponding hyperparameters can affect both the trainability and the generalization of resulting models (Hoffer et al., 2017; Keskar et al., 2017). In this case study, we study the effect of two hyperparameters—learning rate and batch size—that control the effective scale of the noise in stochastic gradient descent (SGD). We study the effect of these hyperparameters in the context of CIFAR-10 (Krizhevsky, 2009) classifiers by comparing the following two learning algorithms:

- **Algorithm** $\mathcal{A}_1$: Training with high SGD noise, i.e., large learning rate (0.1) and small batch size (256). Models attain 93% average test accuracy.
- **Algorithm** $\mathcal{A}_2$: Training with low SGD noise, i.e., small large rate (0.02) and large batch size (1024). Models attain 89% average test accuracy.

Identifying IDTs with MODELDIFF. We apply MODELDIFF just as in Section 4.1, with results shown in Figure 7.

- **Black-and-white texture:** Subpopulation **A** contains two-colored dogs (Figure 7, top right). The low-resolution nature of CIFAR-10 makes it difficult to identify a single distinguishing feature from this subpopulation. Additional analysis using datamodels (Appendix E), however, reveals a set of black-and-white training images from other classes that influence predictions on subpopulation **A** only when models are trained with low SGD noise (algorithm $\mathcal{A}_2$). We thus hypothesize that models trained with low SGD noise rely more on black-and-white textural features to predict the class "dog." To test this hypothesis, we design a transformation that adds a small black-and-white patch to a given image (see Figure 8a).
- **Rectangular shape:** Direction **B** surfaces a subpopulation of front-facing trucks (Figure 7, bottom right) that all have a similar rectangularly-shaped cabin and cargo area. The same style of additional analysis (see Appendix E) supports this observation, and suggests that models trained with low SGD noise (i.e., with $\mathcal{A}_2$) rely on rectangular-shaped patterns to predict the class "truck". To test this hypothesis, we design a feature transformation that modifies a given image with a patch of two high-contrast rectangles, loosely resembling the cabin/cargo shape of trucks in the subpopulation **B** (see Figure 8b).

In Figure 8, we compare the effect of the above feature transformations on models trained with high and low SGD noise. The results again confirm both hypotheses. Adding a small (6px) black-and-white patch to test images increased $P(\text{"dog"})$ by 14% (9%) for models trained with low (high) SGD noise. Similarly, applying a small (8px) rectangular-shape patch increased $P(\text{"truck"})$ by 7% (2%) for models trained with low (high) SGD noise. Increasing the intensity (i.e., patch size) of the transformations again widens the gap in sensitivity between the two model classes.

Connections to prior work. This case study shows how reducing the scale of SGD noise can increase reliance on certain low-level features (e.g., rectangular shape $\rightarrow$ trucks). While prior works show that lower SGD noise worsens aggregate model performance (Keskar et al., 2017; Wen et al., 2019), our methodology identifies *specific* features that are amplified due to low SGD noise. Furthermore, the simplistic nature of the identified features corroborate the theoretical explanation put forth in Li et al. (2019): learning rate scale determines the extent to which models memorize patterns that are easy-to-fit but hard-to-generalize. More broadly, our framework motivates a closer look at how features amplified via low SGD noise alter aggregate model performance.

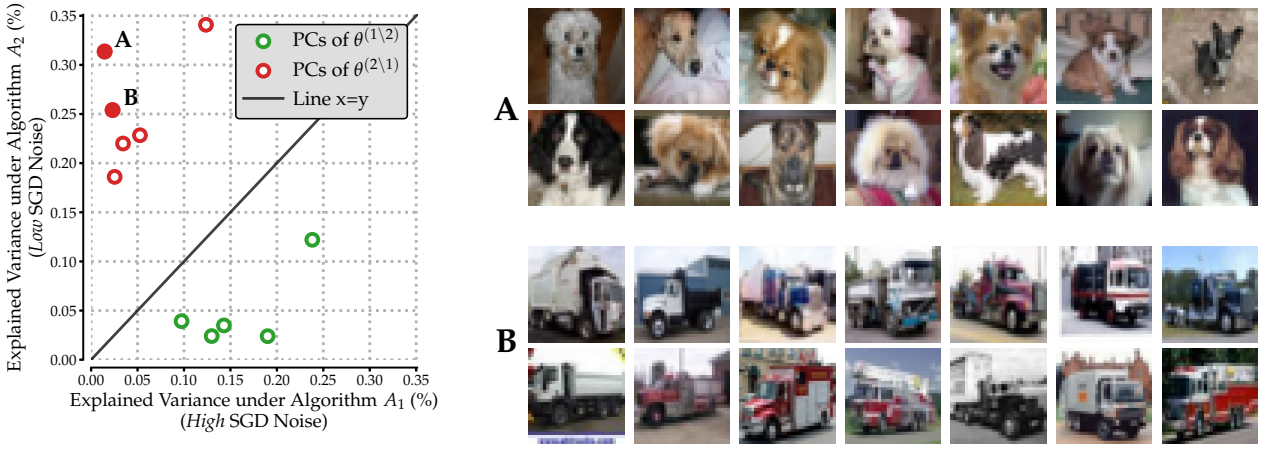
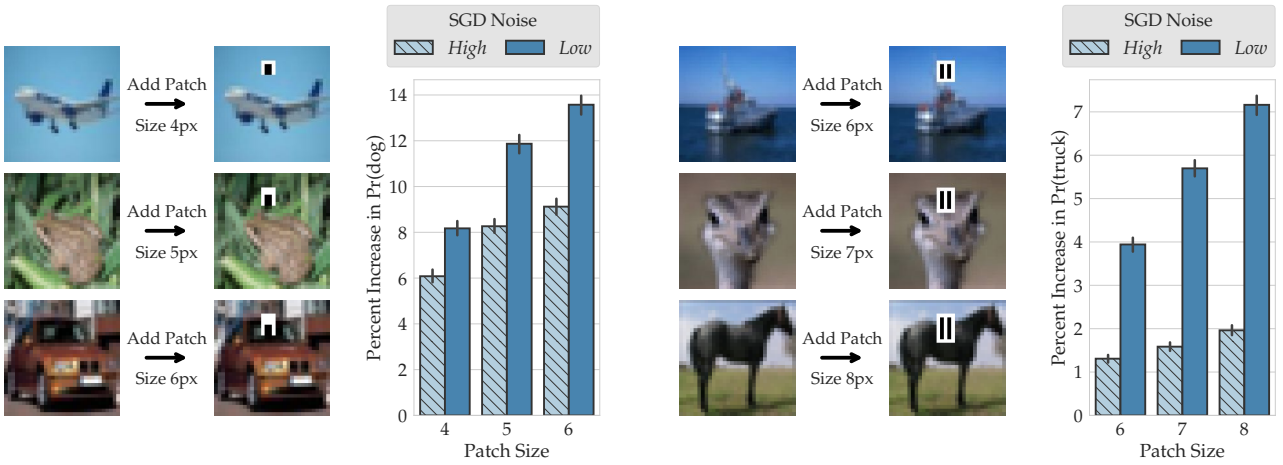


Figure 7: Comparing CIFAR-10 models trained with high and low SGD noise. An analog to Figure 3 for our third case study. Here, subpopulation **A** seems to correspond to dogs with a particular texture, and subpopulation **B** to front-facing trucks with a prominent rectangular shape.



(a) "Black-and-white texture" feature

(b) "Rectangular shape" feature

Figure 8: Effect of SGD hyperparameters on CIFAR-10 models. Analogously to Figures 4 and 6, we use our framework to identify features that distinguish models trained with lower SGD noise from models trained with higher SGD noise. **(Left)** Adding a black-and-white patch to images makes models trained with low (high) SGD noise, on average, 11% (8%) more confident in predicting the label "dog." **(Right)** Adding high-contrast rectangles to images makes models trained with low (high) SGD noise, on average, 5.5% (1.5%) more confident in predicting the label "truck." In both cases, increasing the intensity of feature transformations widens the gap in treatment effect between the two model classes.

C.2. Synthetic spurious correlations

Recall that our algorithm comparison objective (Definition 2.2) counterfactually evaluates distinguishing transformations that we infer via MODELDIFF subpopulations. In this case study, we introduce a semi-synthetic controlled setting in order to *directly* evaluate MODELDIFF subpopulations (i.e., without needing to infer a distinguishing transformation). In particular, we leverage synthetic spurious correlations to plant a "ground-truth" distinguishing subpopulation, which comprises examples on which two algorithms make similar predictions but differ in terms of the underlying learned features. Then, we apply MODELDIFF to this case study, and evaluate the extent to which MODELDIFF recovers the "ground-truth" distinguishing subpopulation.

Dataset. We consider the CIFAR dataset (Krizhevsky, 2009) to construct a binary classification task between CIFAR examples belonging to classes "cat" and "dog". Additionally, as shown in Figure 9, we add a small red square patch to 5% of the examples in the training and test datasets. In particular, 5% of "cat" examples have a red patch in the top left corner, and 5% of "dog" examples have a red patch in the top right corner. In other words, the *location* of the patch—top left corner or top right corner—is fully predictive of the labels "cat" and "dog" respectively.

Learning algorithms. We consider ResNet-9 models trained on the dataset described above with the following algorithms:

- **Algorithm $\mathcal{A}_1$ (models rely on patch)**: Models are trained without any data augmentation and attain 81% average test accuracy. These models latch onto the planted red patch, attaining 100% accuracy on examples containing the patch and 0% when the patch is "flipped" to the other location.
- **Algorithm $\mathcal{A}_2$ (models do not rely on patch)**: Models are trained with *horizontal flip* data augmentation and also attain 81% average test accuracy. Importantly, the horizontal flip augmentation makes the location of the red patch a non-discriminative feature. As a result, these models do not rely on the planted red patch to make predictions. Given an example x, adding a red patch to x on either location (i.e., top left or top right) does not change model predictions on x.

Note that algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ share the same configuration modulo the use of horizontal flips as data augmentation. While models trained with both algorithms attain similar test accuracies, they differ in terms of their reliance on the planted spurious correlation. In particular, on examples containing the red square patch, both model classes make similar predictions, but rely on different "features"—algorithm $\mathcal{A}_1$ outputs models reliant on the patch, whereas $\mathcal{A}_2$ outputs models invariant to the patch. As a result, the set of examples containing the patch correspond to the "ground-truth" distinguishing subpopulation.

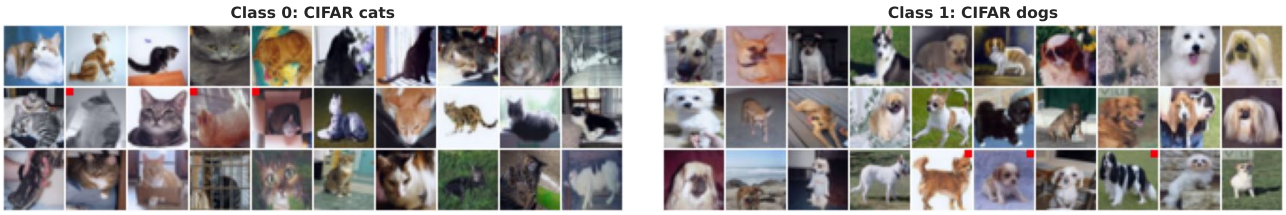


Figure 9: Dataset. Binary classification task between CIFAR "cat" examples (class 0) and CIFAR "dog" examples (class 1). Additionally, 5% of "cat" examples have a (discriminative) red patch in the top left corner, and 5% of "dog" examples have the same patch in the top right corner.

Applying MODELDIFF. Like in previous case studies (Section 4), we apply MODELDIFF to algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ and first obtain a set of distinguishing training directions. In Figure 10, we plot the variance explained by the top few distinguishing training directions in the datamodels for both $\mathcal{A}_1$ (x axis) and $\mathcal{A}_2$ (y axis). In this case, the distinguishing direction corresponding to the "ground-truth" subpopulation should have high explained variance in the datamodels for $\mathcal{A}_1$ and low explained variance in the datamodels for $\mathcal{A}_2$. In Figure 10, we show that the distinguishing subpopulation surfaced by one such distinguishing direction (annotated **A** in Figure 10) surfaces the set of CIFAR examples containing the patch.

Evaluating MODELDIFF subpopulations. Now, we quantitatively assess the extent to which the MODELDIFF subpopulation surfaced via direction **A** (depicted in Figure 10) recovers the "ground-truth" distinguishing subpopulation. More concretely, we compare the similarity between (a) the set of examples containing the patch and (b) the set of top-k examples whose residual datamodels $\theta_i^{(1\setminus 2)}$ most closely aligned with direction **A**. We find that the fraction of top-k examples "retrieved" via MODELDIFF that are in the ground-truth subpopulation equals 1 for all values of k less than the number of examples containing the patch. That is, distinguishing direction **A** (found via MODELDIFF) perfectly recovers the ground-truth distinguishing subpopulation.

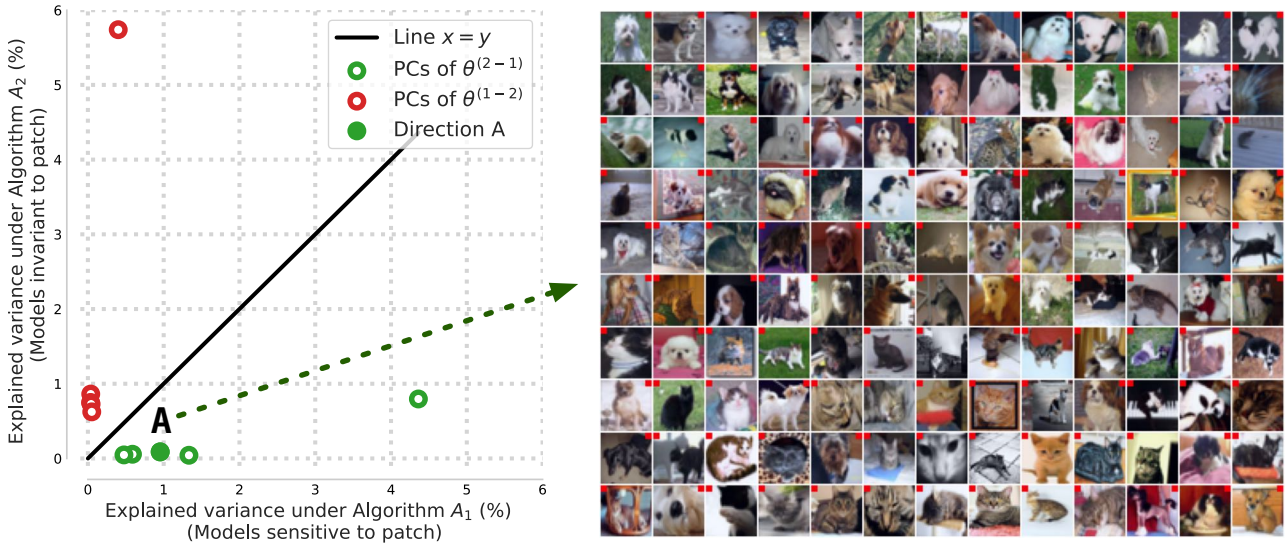


Figure 10: Applying ModelDiff. Comparing algorithms $\mathcal{A}_1$ (models sensitive to patch; no data augmentation) and $\mathcal{A}_2$ (models invariant to patch; horizontal flip augmentation) **(Left)** Each green (resp., red) point is a *training direction* (i.e., a vector $v \in \mathbb{R}^{|S|}$ representing a weighted combination of training examples) that distinguishes $\mathcal{A}_1$ from $\mathcal{A}_2$ (resp., $\mathcal{A}_2$ from $\mathcal{A}_1$) as identified by MODELDIFF (i.e., each point is a solution to (4)). The x and y coordinates of each point represent the "importance" (as given by Proposition 3.1) of the training direction to models trained with $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. **(Right)** The distinguishing subpopulation corresponding to the distinguishing direction **A**. This subpopulation (extracted via MODELDIFF extracts *all* examples (with patch) in the "ground-truth' distinguishing subpopulation.

D. Extending MODELDIFF

D.1. Aggregate metric for algorithm comparison

We can repurpose our framework as a similarity metric that quantifies the similarity of models trained with different learning algorithms in a more global manner. A straightforward approach to output a similarity score (or distribution) is to compute the cosine similarity of datamodel vectors. More concretely, let $\theta_i^{(1)}$ and $\theta_i^{(2)}$ denote the datamodels of example x_i with respect to models trained using learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$. Then, the cosine similarity between $\theta_i^{(1)}$ and $\theta_i^{(2)}$ measures the extent to which models trained with $\mathcal{A}_1$ and $\mathcal{A}_2$ depend on the same set of training examples to make predictions on example x_i.

We apply this metric to two case studies—pre-training (WATERBIRDS) and SGD noise (CIFAR-10)—in Figure 11. Specifically, Figure 11 plots the distribution of cosine similarity of datamodels for multiple learning algorithms over all test examples. The left subplot shows that on WATERBIRDS, ImageNet-pretrained ResNet50 models are, on average, more similar to ImageNet-pretrained ResNet18 models than to ResNet50 models pre-trained on synthetically generated data (Baradad Jurjo et al., 2021) and models trained from scratch. The right subplot shows that on CIFAR-10, ResNet9 models trained with high SGD noise are more similar to smaller-width ResNet9 models trained with high SGD noise than to ResNet9 models trained with low SGD noise.

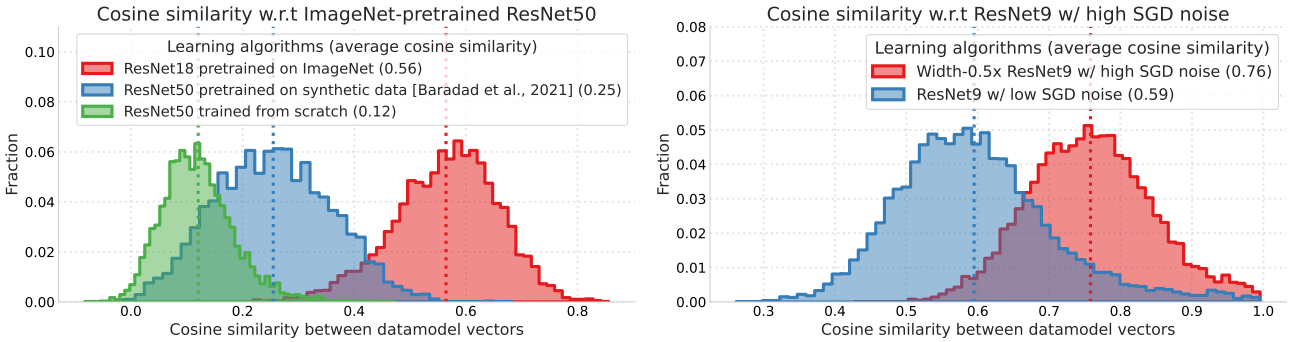


Figure 11: Datamodel cosine similarity. We use cosine similarity between two datamodel vectors as an aggregate metric to quantify the similarity of models trained with different learning algorithms. **(Left)** On WATERBIRDS data, datamodels of ImageNet-pretrained ResNet50 and ResNet18 models are more similar to each other than to models pre-trained on synthetically generated data and models trained from scratch. **(Right)** On CIFAR-10 data, ResNet models trained with high SGD noise are more similar to each other to ResNet models trained with low SGD noise.

D.2. Leveraging CLIP to analyze distinguishing subpopulations

As discussed in Section 3 and Appendix E, we infer distinguishing features candidates through manual inspection of distinguishing subpopulations. In this section, we demonstrate that for image classifiers, shared vision-language models such as CLIP (Radford et al., 2021) provide a streamlined alternative to manual inspection of distinguishing subpopulations.

Approach. Before we describe our approach, note that CLIP is a contrastive learning method that embeds text and natural language into a shared embedding space. Our approach leverages CLIP embeddings to identify multiple *distinguishing captions*—representative descriptions that best contrast a given subpopulation of images from a set of images sampled from the same distribution. In the context of our framework, the CLIP-based approach takes as inputs a distinguishing training direction v, a set of images $\mathcal{D}$, and a set of captions $\mathcal{S}$[8], and outputs a set of distinguishing captions $\mathcal{S}' \in \mathcal{S}$ in four steps:

- *Pre-compute image and text embeddings*. Use the image encoder of a CLIP model to compute a set of normalized embeddings for all images in $\mathcal{D}$. Analogously, use the text encoder of a CLIP model to compute a set of normalized embeddings for all captions in $\mathcal{S}$.
- *Record image-text pairwise cosine similarity*. Let vector $C_i \in \mathcal{R}^{|\mathcal{S}|}$ denote the pairwise cosine similarity between the embedding of image $i \in \mathcal{D}$ and all captions $j \in \mathcal{S}$.
- *Compute mean cosine similarity over dataset and top-k subpopulation*. Compute the mean cosine similarity vector $\bar{C} = \frac{1}{n}\sum_{i \in \mathcal{D}} C_i$ over all images in $\mathcal{D}$. Similarly, given distinguishing training direction v, compute the mean cosine similarity vector $C^{(v)}$ over the top-k images whose residual datamodel vectors are most aligned with v.
- *Extract distinguishing captions $\mathcal{S}'$*. Use cosine similarity vectors $\bar{C}$ and $C^{(v)}$ to extract captions in $\mathcal{S}$ that have the maximum difference between $C_i^{(v)}$ and $\bar{C}_i$.

Intuitively, the set of distinguishing captions $\mathcal{S}'$ correspond to representative captions (or, descriptions) that best contrast the top-k images surfaced by distinguishing direction v from the dataset.

Results. We now apply this approach to our case study on ImageNet pre-training, where we compare WATERBIRDS models trained with and without ImageNet pre-training (see Section 4). Specifically, we evaluate whether the CLIP-based approach surfaces distinguishing captions that are similar to distinguishing features "yellow color" (direction **A**) and "human face" (direction **B**) inferred via manual inspection. Figure 12 illustrates that for direction **A**), the CLIP-based approach highlights distinguishing captions such as `yellow`, `lemon`, and `sulphur`, all of which are similar to the "yellow color" feature that we infer via manual inspection. Similarly, Figure 13 shows that the distinguishing captions for direction **B** (e.g., `florist`, `faces`, `counselors`) are similar to the identified "human face" feature.

To summarize, we show how the verification step can be easily *specialized* to comparisons of vision classifiers trained on ImageNet-like data via vision-language embeddings such as CLIP.

[8]We use a filtered list of roughly 20,000 most common English words in order of frequency, taken from https://github.com/first20hours/google-10000-english.

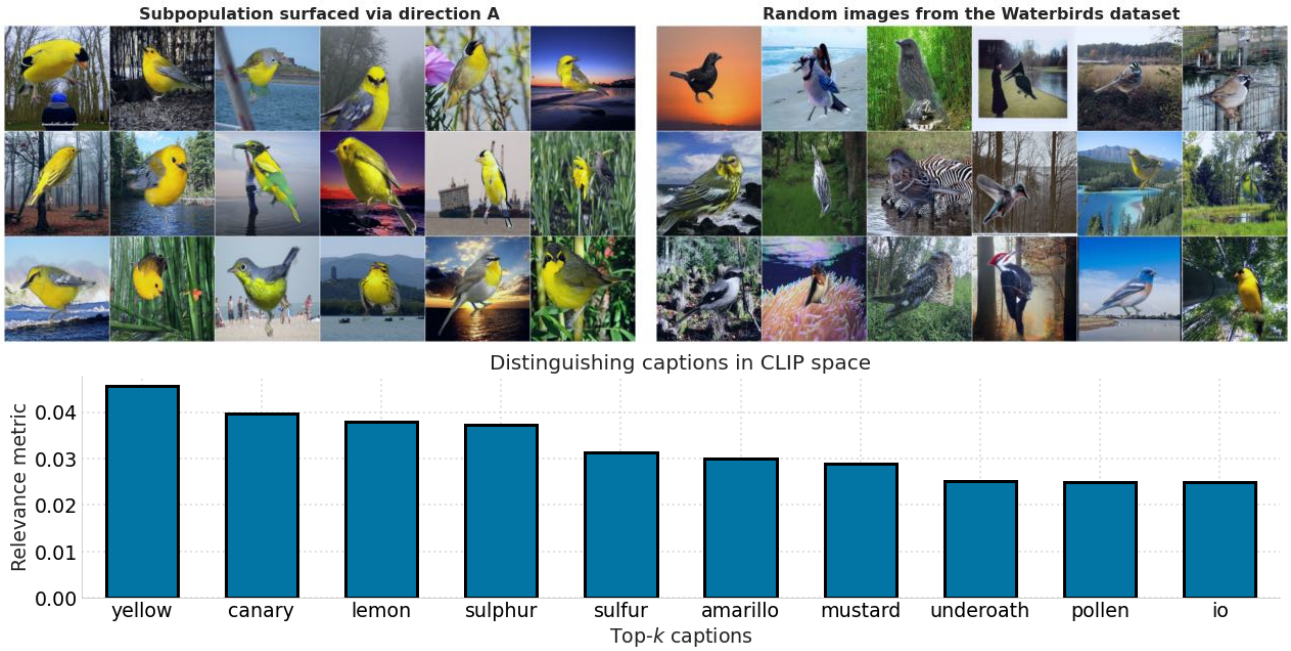


Figure 12: Direction A. The CLIP-based approach extracts distinguishing captions such as `yellow`, `lemon`, and `sulphur`, all of which contrast the residual subpopulation on the left to a set of random images from the WATERBIRDS dataset on the right. These distinguishing captions match the "yellow color" feature that we infer via manual inspection of the distinguishing subpopulation in Section 4.2.

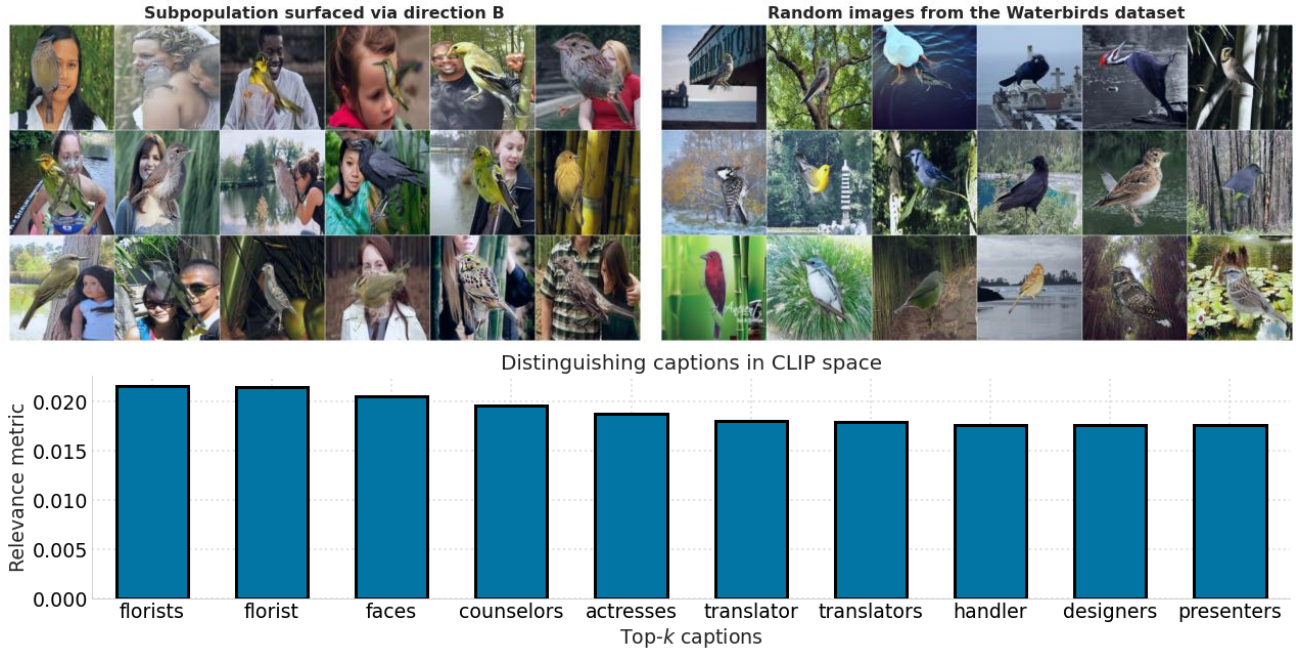


Figure 13: Direction B. The CLIP-based approach extracts distinguishing captions such as `florists`, `faces`, and `counselors`, all of which contrast the residual subpopulation (left) of images with human faces in the background to a set of random images (right) from the WATERBIRDS dataset. These distinguishing captions match the "human face" feature that we infer and counterfactually verify via manual inspection of the distinguishing subpopulation in Section 4.2.

E. Additional analysis of distinguishing subpopulations

As outlined in Section 3, we analyze distinguishing subpopulations to infer (and test) distinguishing feature transformations. In this section, we present additional analysis in order to substantiate the distinguishing features inferred in each case study.

First, we describe two additional tools that we use to analyze subpopulations surfaced by principal components (PCs) of residual datamodels.

- **Class-specific visual inspection.** As shown in Section 4, the subpopulation of test examples whose datamodels have maximum projection onto PCs of residual datamodels largely belong to same class; these subpopulation mostly surface images from the same class. So, a simple-yet-effective way to identify *subpopulation-specific* distinguishing feature(s) is to just visually contrast the surfaced subpopulation from a set of randomly sampled examples that belong to the same class.

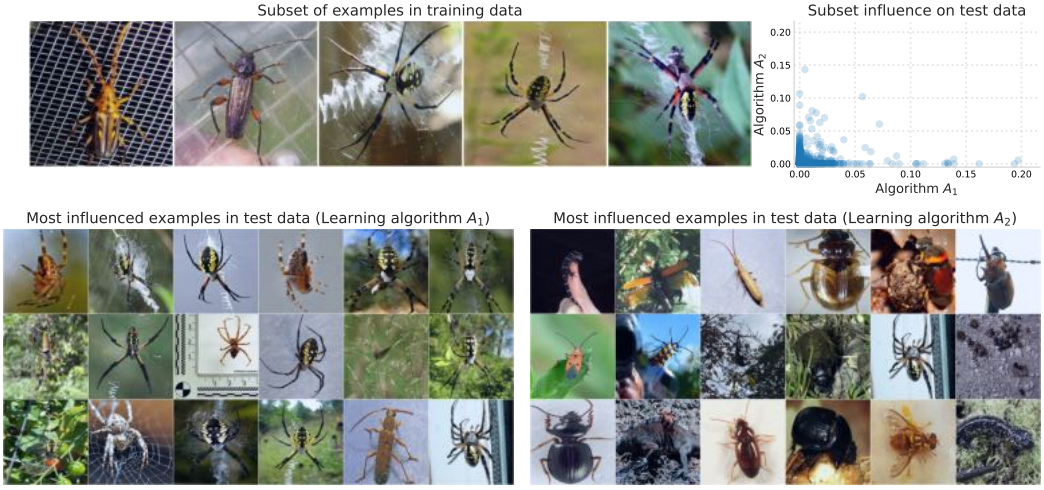
- **Relative influence of training examples.** Given a subset of test examples $S' \subset S$, can we identify a set of training examples $T' \subset T$ that strongly influence predictions on S' when models are trained with algorithm $\mathcal{A}_1$ but not when trained with $\mathcal{A}_2$? Given datamodel representations $\{\theta_i^{(1)}\}$ for $\mathcal{A}_1$ and $\{\theta_i^{(1)}\}$ for $\mathcal{A}_2$, we apply a two-step (heuristic) approach identify training examples with high influence on $\mathcal{A}_1$ relative to $\mathcal{A}_2$:

  – First, given learning algorithm $\mathcal{A}_i$ and test subset S', we estimate the aggregate (positive or negative) influence of training example x_k on subset S by taking the absolute sum over the corresponding datamodel weights: $\sum_{j \in S'} |\theta_{jk}^{(i)}|$.
  – Then, we take the absolute difference between the aggregate influence estimates of training example x_k using $\theta^{(1)}$ and $\theta^{(2)}$. This difference measures the *relative influence* of training example x_k on predictions of test subset S when models are trained with algorithm $\mathcal{A}_1$ instead of algorithm $\mathcal{A}_2$.

  In our analysis, we (a) identify training examples that have top-most relative influence estimates and then (b) visually contrast the subsets of test examples (one for each learning algorithm) that are most influenced by these training examples.

E.1. Case study: Standard data augmentation

Our case study on LIVING17 data in Section 4.1 shows that standard data augmentation can amplify co-occurrence bias (spider web) and texture bias (polka dots). We further substantiate these findings with relative influence analysis (Figure 14) and class-specific visual inspection (Figure 17).



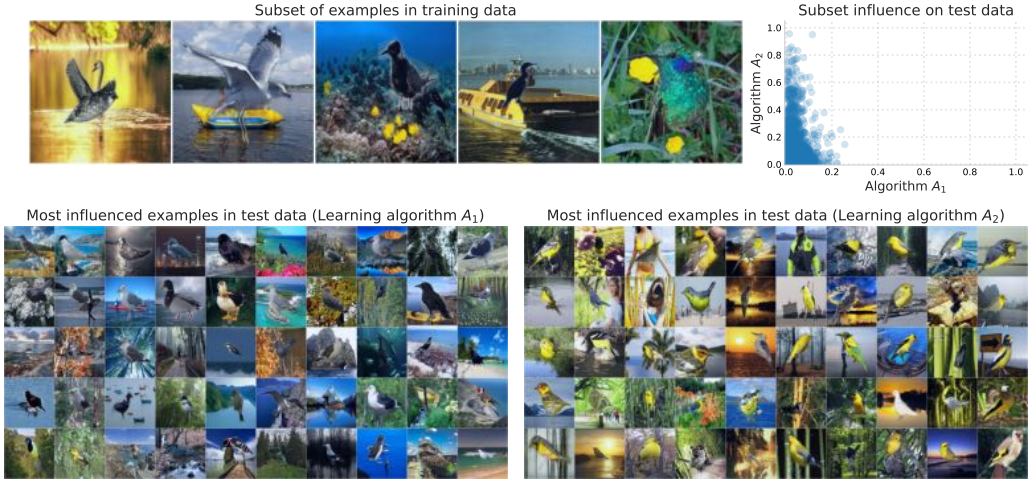
(a) "Spider web" feature

(b) "Polka dots" feature

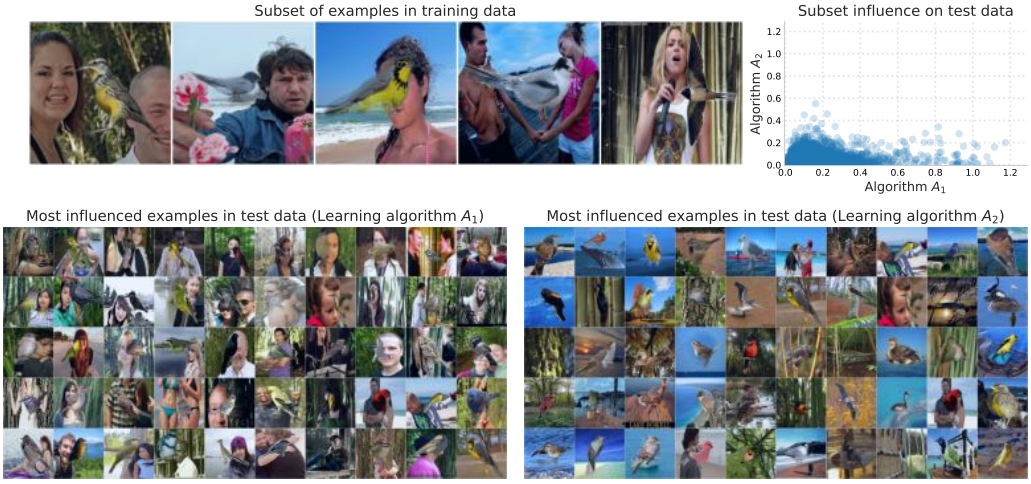
Figure 14: Relative influence of training data on LIVING17 subpopulations. **Panel (a)**: Training images that contain web-like patterns have high relative influence on the "spider web" test subpopulation (see Figure 3). These images strongly influence model predictions on test images that contain spider webs (in bottom row) only when models are trained with augmentation (algorithm $\mathcal{A}_1$). **Panel (b)**: Training images that contain yellow-black texture have high relative influence on the "polka dots" test subpopulation (see Figure 3). These images strongly influence model predictions on test images of salamanders with yellow polka dots (in bottom row) only when models are trained with augmentation (algorithm $\mathcal{A}_1$).

E.2. Case study: ImageNet pre-training

Our case study on WATERBIRDS data shows that ImageNet pre-training reduces dependence on the "yellow color" feature, but introduces dependence the "human face" feature. We support these findings with relative influence analysis in Figure 15 and additional inspection in Figure 18.



(a) "Yellow color" feature

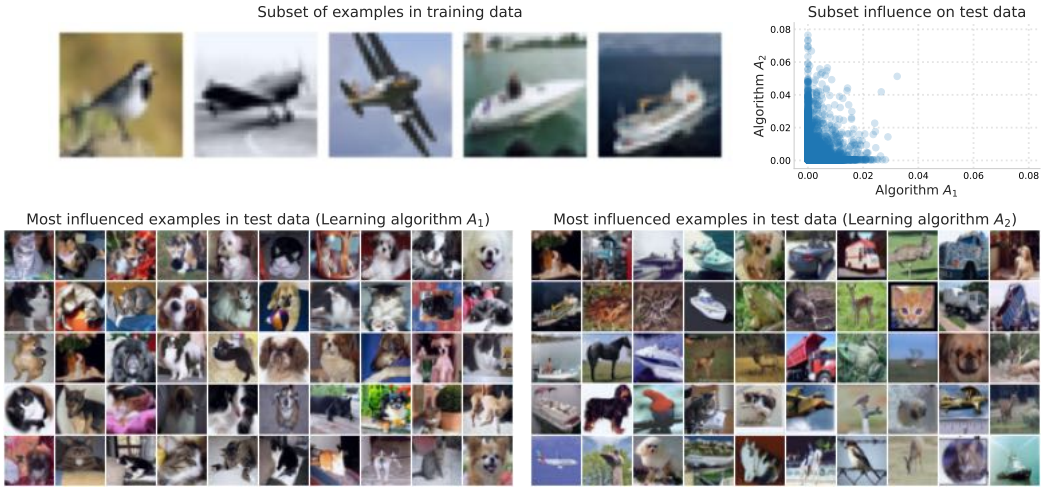


(b) "Human face" feature

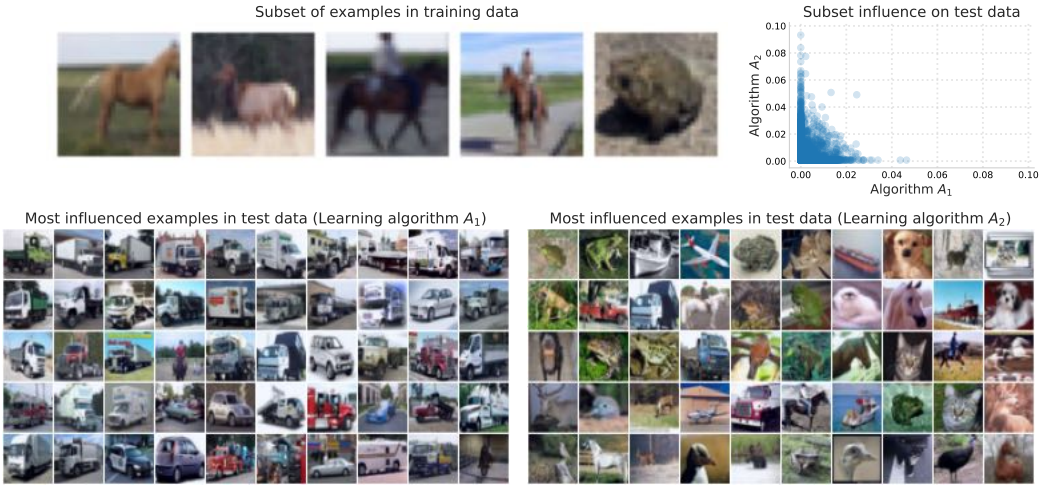
Figure 15: Relative influence of training data on WATERBIRDS subpopulations. **Panel (a)**: Training images with yellow objects in the background have high relative influence on the "yellow color" test subpopulation (see Figure 5). These images strongly influence model predictions on test images that have yellow birds / objects (bottom row) only when models are trained from scratch (algorithm $\mathcal{A}_2$). **Panel (b)**: Training images that contain human faces in the background have high relative influence on the "human face" test subpopulation (see Figure 5). These images strongly influence model predictions on test images (in bottom row) with human face(s) only when models are pre-trained on ImageNet (algorithm $\mathcal{A}_1$).

E.3. Case study: SGD hyperparameters

We analyze relative influence (Figure 16), and class-specific subpopulations (Figure 19) to hone in on two instances of distinguishing features–black-and-white texture and rectangular shape— in CIFAR-10 data that are amplified by low SGD noise.



(a) "Black-white texture" bias



(b) "Rectangular shape" bias

Figure 16: Relative influence of training data on CIFAR-10 subpopulations. **Panel (a)**: Training images with black-white objects have high relative influence on the "black-white" dog subpopulation (see Figure 7). These images influence model predictions on test images of black-white dogs (in bottom row) only when models are trained with low SGD noise (alg. $\mathcal{A}_2$). **Panel (b)**: Training images with high-contrast rectangular components in the background have high relative influence on the "rectangular shape" truck subpopulation (see Figure 7). These images influence model predictions on test images of front-facing trucks with prominent rectangular components (in bottom row) only when models are trained with low SGD noise (alg. $\mathcal{A}_2$).

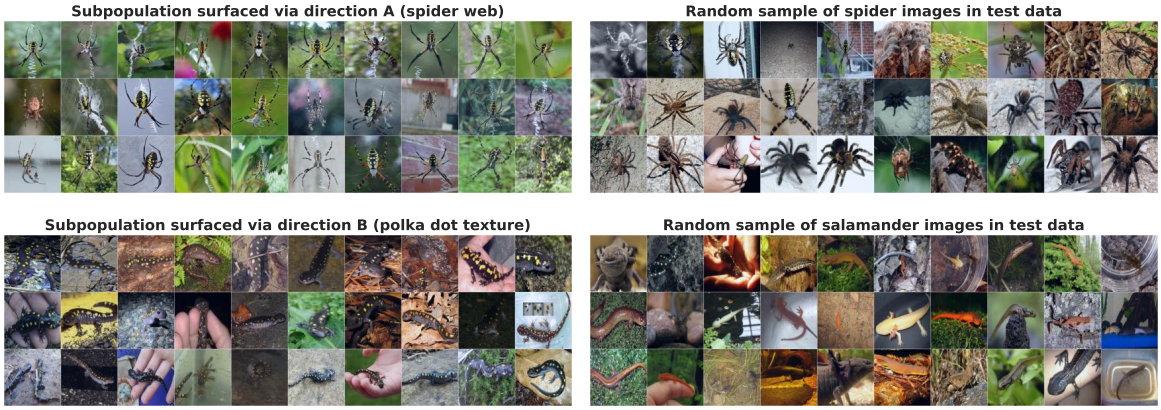


Figure 17: Class-specific visual inspection of LIVING17 subpopulations. **(Top)** In contrast to random LIVING17 images of spiders, the "spider web" subpopulation surfaces spiders with a prominent spider web in the background. **(Bottom)** Unlike random LIVING17 images of salamanders, the "polka dots" subpopulation surfaces salamanders that have a yellow-black polka dot texture.

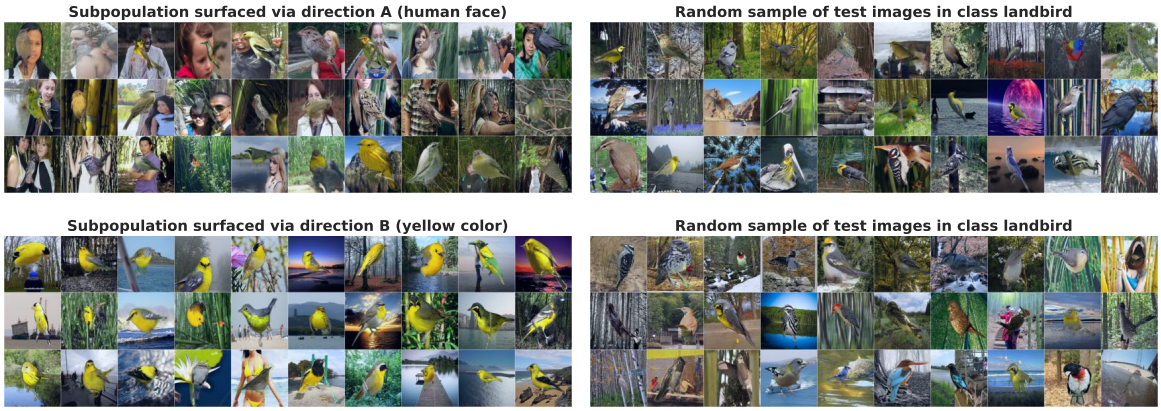


Figure 18: Class-specific visual inspection of WATERBIRDS subpopulations. **(Top)** In contrast to random "landbird" images, the "human face" subpopulation surfaces landbirds with human face(s) in the background. **(Bottom)** Unlike random "landbird" images, the "yellow color" subpopulation surfaces images with yellow birds *or* yellow objects in the background.

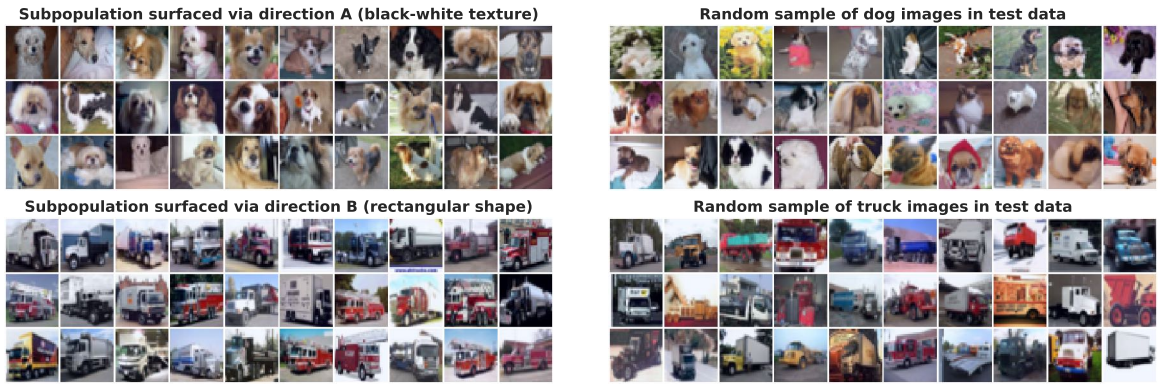


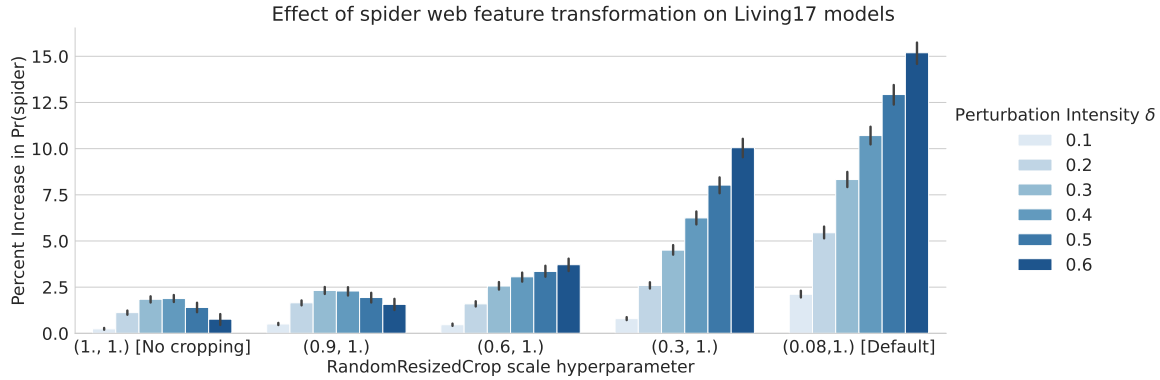
Figure 19: Class-specific visual inspection of CIFAR-10 subpopulations. In contrast to random images of dogs (top) and trucks (bottom), the "black-white" and "rectangular shape" subpopulations surface images of black-white dogs and front-facing trucks with multiple rectangular components respectively.

F. Additional evaluation of distinguishing transformations

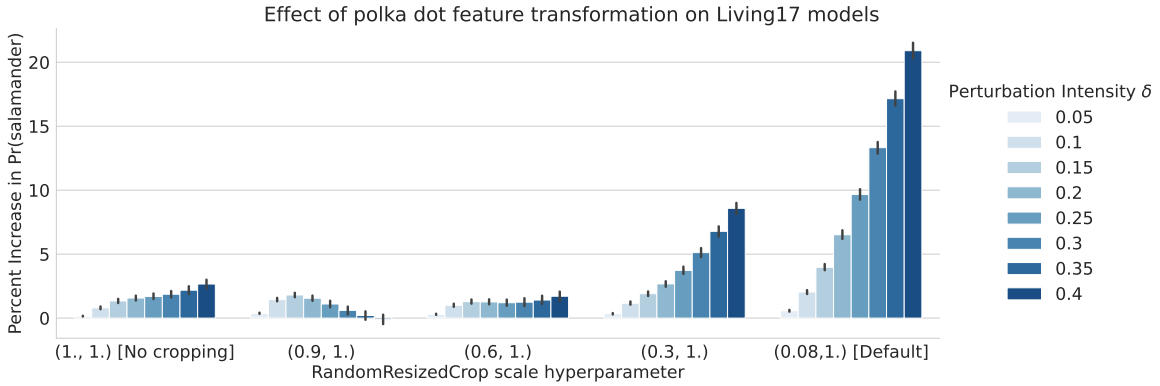
Distinguishing feature transformations, which we recall from Section 3, are functions that, when applied to data points, change the predictions of one model class—but not the other—in a consistent way. In our case studies, we design distinguishing feature transformations that counterfactually verify features inferred from distinguishing subpopulations. Our findings in Section 4 use feature transformations to quantitatively measure the relative effect of the identified features on models trained with different learning algorithms. In this section, we present additional findings on feature transformations for each case study:

F.1. Case study: Standard data augmentation

In Section 4.1, we showed that standard data augmentation—horizontal flips and random crops—amplifies LIVING17 models' reliance on "spider web" and "polka dots" to predict spiders and salamanders respectively. Figure 20 verifies our findings over a larger range of perturbation intensity δ values. We also observe that decreasing the minimum allowable crop size in `RandomResizedCrop` from 1.0 (i.e., no random cropping) to 0.08 (default `torchvision` hyperparameter) increases models' sensitivity to both feature transformations.



(a) "Spider web" feature

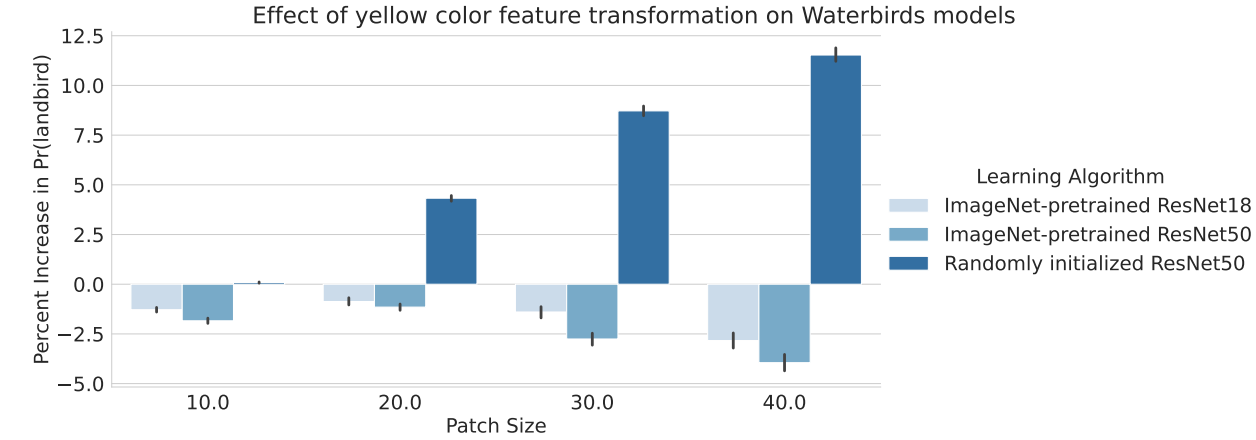


(b) "Polka dots" feature

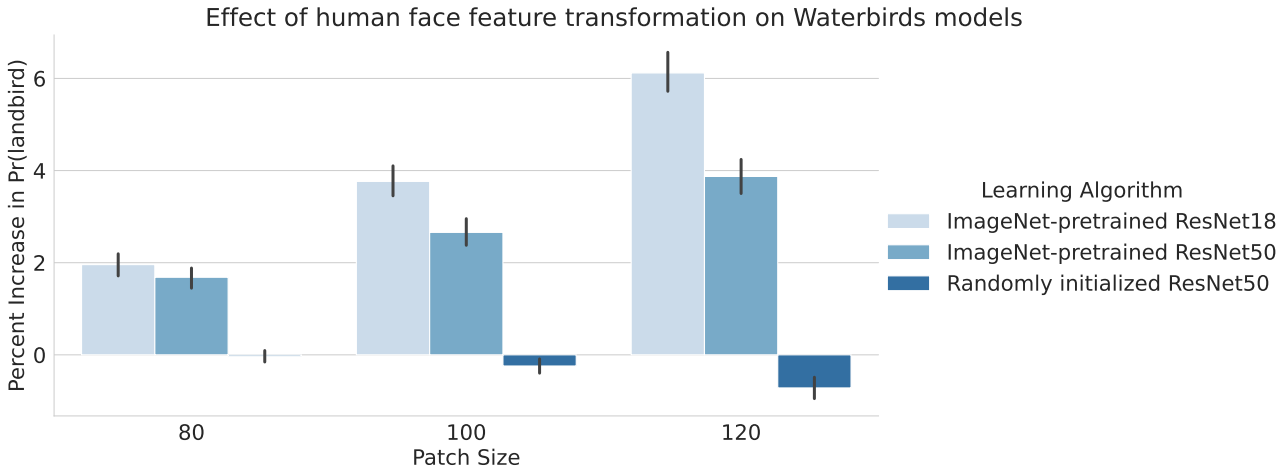
Figure 20: Additional evaluation of LIVING17 feature transformations. The top and bottom row evaluate the effect of "spider web" and "polka dot" feature transformations on models trained with different data augmentation schemes. Increasing the intensity of the transformations and the minimum crop size of `RandomResizedCrop` augmentation (via `scale` hyperparameter) increases the sensitivity of models to both feature transformations in a consistent manner.

F.2. Case study: ImageNet pre-training

In Section 4.2, we showed that fine-tuning ImageNet-pretrained ResNet50 models on WATERBIRDS data instead of training from scratch alters the relative importance of two spurious features: "yellow color" and "human face". In Figure 21, we show that both feature transformations alter the predictions of ImageNet-pretrained ResNet18 and ImageNet-pretrained ResNet50 models in a similar way.



(a) "Yellow color" feature

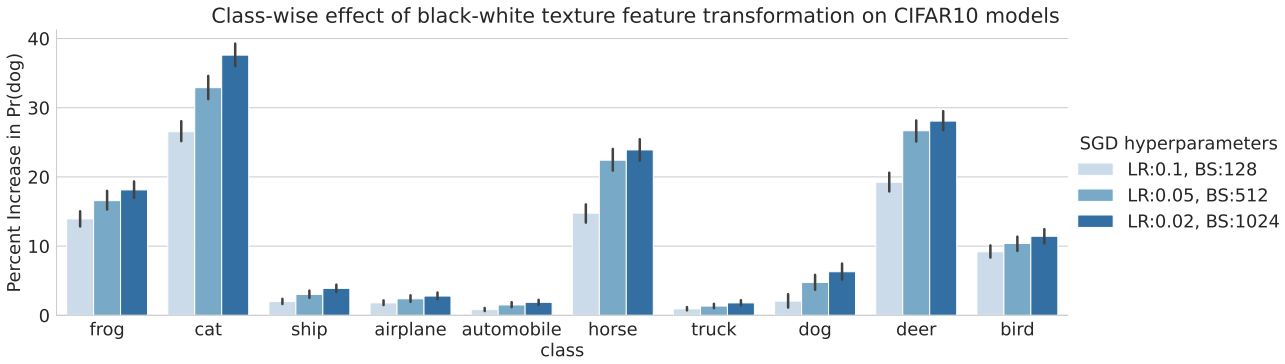


(b) "Human face" feature

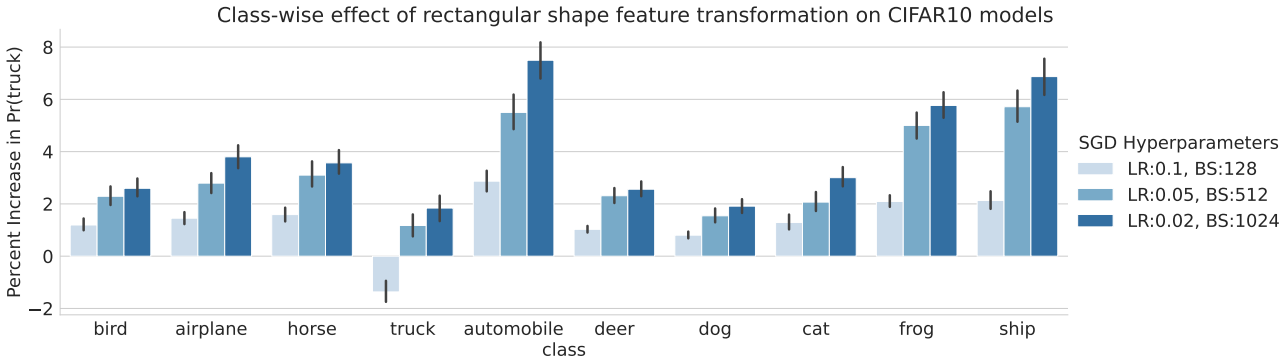
Figure 21: Additional evaluation of WATERBIRDS feature transformations. The top and bottom row evaluate the effect of "yellow color" and "human face" feature transformations on models trained with and without ImageNet pre-training. In both cases, unlike ResNet50 models trained from scratch, ImageNet-pretrained ResNet18 and ResNet50 models are sensitive to the "human face" transformation but not to the "yellow color" transformation.

F.3. Case study: SGD hyperparameters

In Appendix C.1, we showed that reducing SGD noise results in CIFAR-10 models that are more sensitive to certain features, such as rectangular shape bias and black-white texture to predict trucks and dogs. In Figure 22, we evaluate how feature transformations change class-wise predictions of models trained with different SGD learning rate and batch size hyperparameters.



(a) "Black-white texture" feature



(b) "Rectangular shape" feature

Figure 22: Additional evaluation of CIFAR-10 feature transformations. The top and bottom row evaluate the effect of "black-white texture" and "rectangular shape" feature transformations on CIFAR-10 models trained with high (light blue), medium, and low (dark blue) SGD noise. In both cases, models trained with higher SGD noise are, on average, more sensitive to these transformations across all classes. Furthermore, the effect of the transformations are class-dependent—model predictions on transformed examples from semantically similar classes differ to a greater extent.

G. Miscellaneous results

G.1. Explained variance of residual datamodel principal components

Recall from Section 3 that the fraction of variance in datamodel representations $\{\theta_x^{(i)}\}$ explained by training direction v signifies the importance of the direction (or, combination of training examples) to predictions of models trained with algorithm $\mathcal{A}_i$. Through our case studies in Section 4, we show that the top $5-6$ principal components (PCs) of residual datamodels $\theta^{(1\setminus 2)}$ correspond to training directions that have high explained w.r.t. datamodels of algorithm $\mathcal{A}_1$ but not $\mathcal{A}_2$, and vice versa. Figure 23 shows that the top-100 PCs of residual datamodel $\theta^{(1\setminus 2)}$ (resp., $\theta^{(2\setminus 1)}$) have more (resp., less) explained variance on datamodel $\theta^{(1)}$ than on datamodel $\theta^{(2)}$.

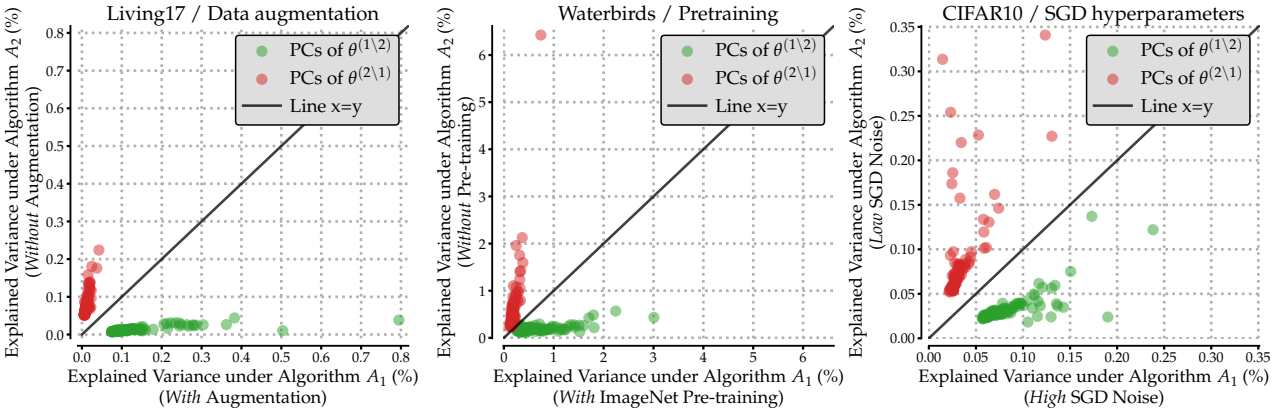


Figure 23: Explained variance of residual datamodels' principal components. Highlighted in green (resp. red), the top-100 PCs of residual datamodel $\theta^{(1\setminus 2)}$ (resp. $\theta^{(2\setminus 1)}$) explain a larger (resp. smaller) fraction of datamodel variance under algorithm $\mathcal{A}_1$ than under algorithm $\mathcal{A}_2$ across all three case studies.

G.2. Effect of sample size on datamodel estimation

In this section, we analyze the effect of sample size on datamodel estimation.

Setup. Recall from Appendix B.3 that a datamodel training set of size m corresponds to training m models on independently sampled training data subsets. For our case study on ImageNet pre-training in Section 4.2, we estimate datamodels on WATERBIRDS data with $50,000$ samples (i.e., m ResNet50 models trained on random subsets of the WATERBIRDS training dataset). In this experiment, we analyze how the estimated datamodels vary as a function of sample size $m \in \{5000, 10000, 25000, 50000\}$.

Cosine similarity between datamodels. Our algorithm comparisons framework uses normalized datamodel representations to compute distinguishing training directions in the first step. So, we first analyze the alignment between datamodel representations that are estimated with different sample sizes. Specifically, we evaluate the cosine similarity between $\theta_x^{(m_1)}$ and $\theta_x^{(m_2)}$, where vector $\theta_x^{(m)} \in \mathbb{R}^{|S|}$ corresponds to the linear datamodel for example x estimated with m samples. As shown in Figure 24, the average cosine similarity between datamodels is greater than 0.9 even when the sample size is reduced by a factor of 10, from 50000 to 5000.

Explained variance of principal components. As discussed in Section 4, for a given training direction v, the fraction of variance that v explains in datamodel representations $\{\theta_x^{(i)}\}$ captures the importance of direction v (i.e., weighted combination of training examples) to the predictions of models trained using algorithm $\mathcal{A}_i$. Here, we show that principal components of datamodel representations trained with smaller sample size (e.g., $m = 5000$) have similar explained variance on datamodel representations estimated with larger sample size, and vice-versa. As shown in Figure 25, the explained variance of the top-10 principal components of datamodels estimated with $m \in \{5000, 50000\}$ have similar explained variance on datamodels estimated with $m \in \{5000, 10000, 25000, 50000\}$.

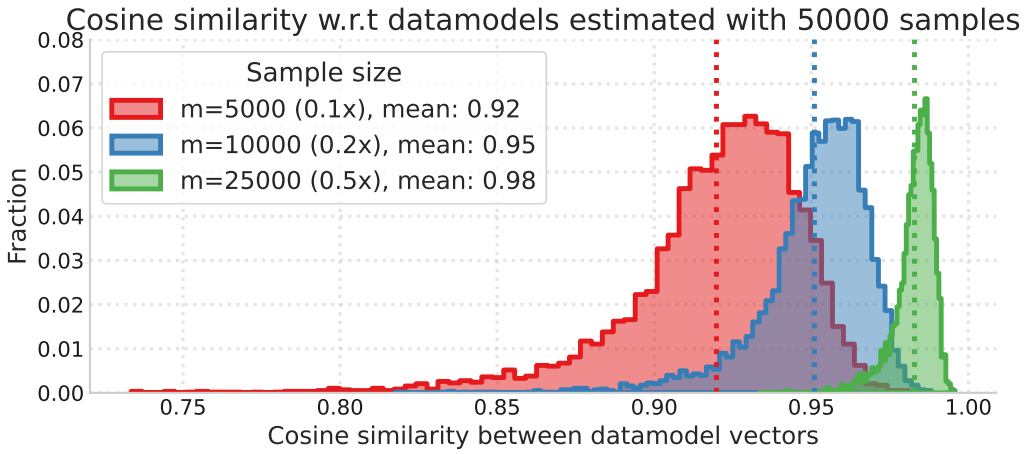


Figure 24: Histogram over cosine similarity between datamodels $\theta_x^{(m_1)}$ and $\theta_x^{(m_2)}$, where vector $\theta_x^{(m)} \in \mathbb{R}^{|S|}$ corresponds to the linear datamodel for example x estimated with $m \in \{5000, 10000, 25000, 5000\}$ samples.

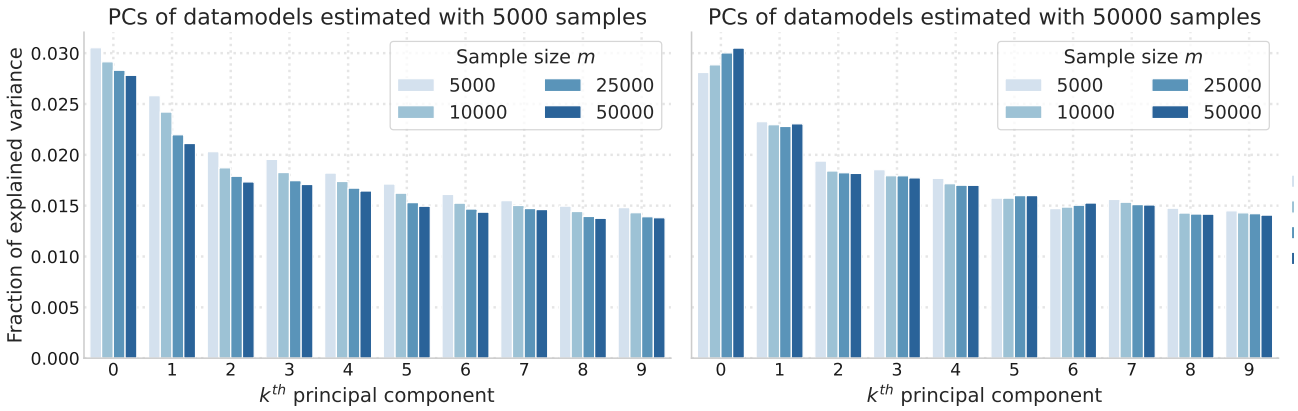


Figure 25: Explained variance of the top-10 principal components of datamodels estimated with $m \in \{5000, 50000\}$ have similar explained variance on datamodels estimated with sample size $m \in \{5000, 10000, 25000, 50000\}$.

G.3. Algorithm comparisons with penultimate-layer representations

In this section, we contrast our algorithm comparisons framework to comparisons based on model predictions and penultimate-layer representations. Note that there are no existing methods that can be directly reused for comparing learning algorithms to the best of our knowledge. Therefore, we design experiments to evaluate whether model predictions and penultimate-layer representations can identify distinguishing training directions surfaced using our framework.

Model predictions. Through this experiment, we show that example-level differences in predictions (Zhong et al., 2021; Meding et al., 2022) of models trained with different algorithms are not necessary to identify subpopulations analysed in our case studies. First, we re-run the first stage of our framework only on test examples on which models trained with different algorithm have the same prediction on average. Then, we compare distinguishing training directions (i.e., output of the first stage) before and after controlling for prediction-level agreement. Our results in Table 1 show that for each case study, our framework identifies similar training directions (i.e., high cosine similarity) even after removing test examples on which model predictions differ. This experiment shows that our framework can identify fine-grained differences between learning algorithms that persist even after controlling for prediction-level disagreement across models trained with different algorithms.

Dataset / Case study	Direction	(Absolute) Cosine Similarity
Living17 / Data augmentation	**A** (Spider web)	0.999
	B (Polka dots)	0.998
Waterbirds / ImageNet pre-training	**A** (Yellow color)	0.977
	B (Human face)	0.740
CIFAR-10 / SGD hyperparameters	**A** (Black-white texture)	0.998
	B (Rectangular shape)	0.999

Table 2: Distinguishing training directions before and after filtering out high-disagreement test examples exhibit high cosine similarity and surface subpopulations of images that share the same distinguishing feature.

Penultimate-layer representations. Representation-based comparisons (Raghu et al., 2017; Morcos et al., 2018b; Kornblith et al., 2019) measure the degree to which different models' representation can be aligned. Unlike datamodel representations, penultimate-layer representations are not aligned—coordinates of penultimate-layer representations do not share a consistent interpretation across different models. So, we first introduce a variant based on penultimate-layer representations that has a consistent basis. Specifically, similar to how the datamodel weight θ_{ij} denotes the influence of training example j over the prediction on test example i, we set $\theta^{(r)}_{ij}$ to equal the cosine similarity between the penultimate-layer representation of test example i and train example j. We then compare two properties of datamodel representations and penultimate-layer representations:

- **Effective dimensionality of representations**: We first compare the effective dimensionality (i.e., cumulative fraction of variance explained by top-k components) of datamodel representations θ and penultimate-layer representations $\theta^{(r)}$. Figure 26 shows that for all datasets and learning algorithms, datamodel representations have significantly higher effective dimensionality than the penultimate-layer alternative. For example, on CIFAR-10 data, more than 99% of the variation in penultimate-layer representations is captured by the first 10 components.
- **Explained variance of distinguishing training directions**: We now re-run the first stage of our framework to compare distinguishing directions obtained via datamodel representations θ and penultimate-layer representations $\theta^{(r)}$. Here, we evaluate the extent to which these training directions distinguish models trained with different algorithms. Specifically, as shown in Figure 27, we compare the difference in the cumulative fraction of variance explained by the top-k training directions across representations corresponding to algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ (higher the better). Figure 27 shows that (a) training directions obtained from datamodel representations have significantly higher gap in explained variance across learning algorithms and (b) directions obtained from penultimate-layer representations can have close to zero or negative gap in explained variance across learning algorithms.

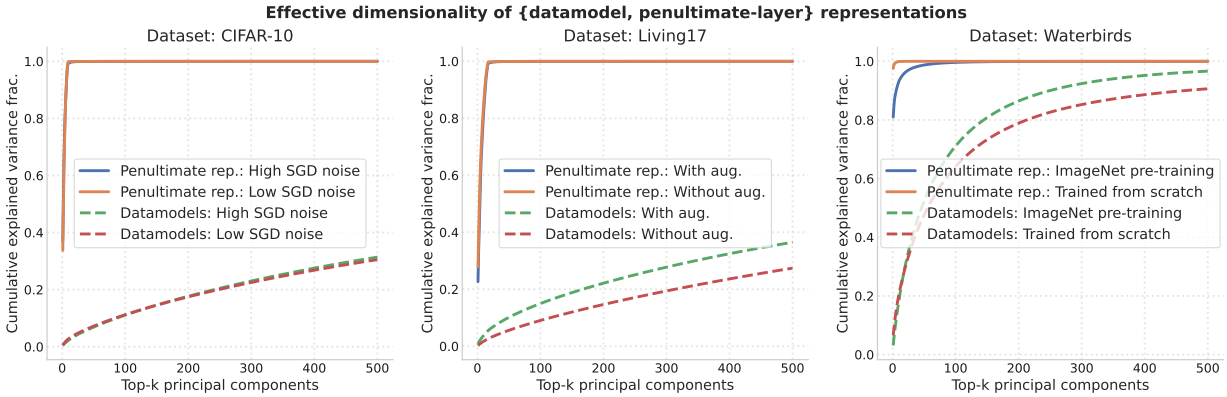


Figure 26: Effective dimensionality (i.e., cumulative fraction of variance explained by top-k components) of datamodel representations is significantly more than that of penultimate-layer representations across all datasets and learning algorithms considered in Section 4.

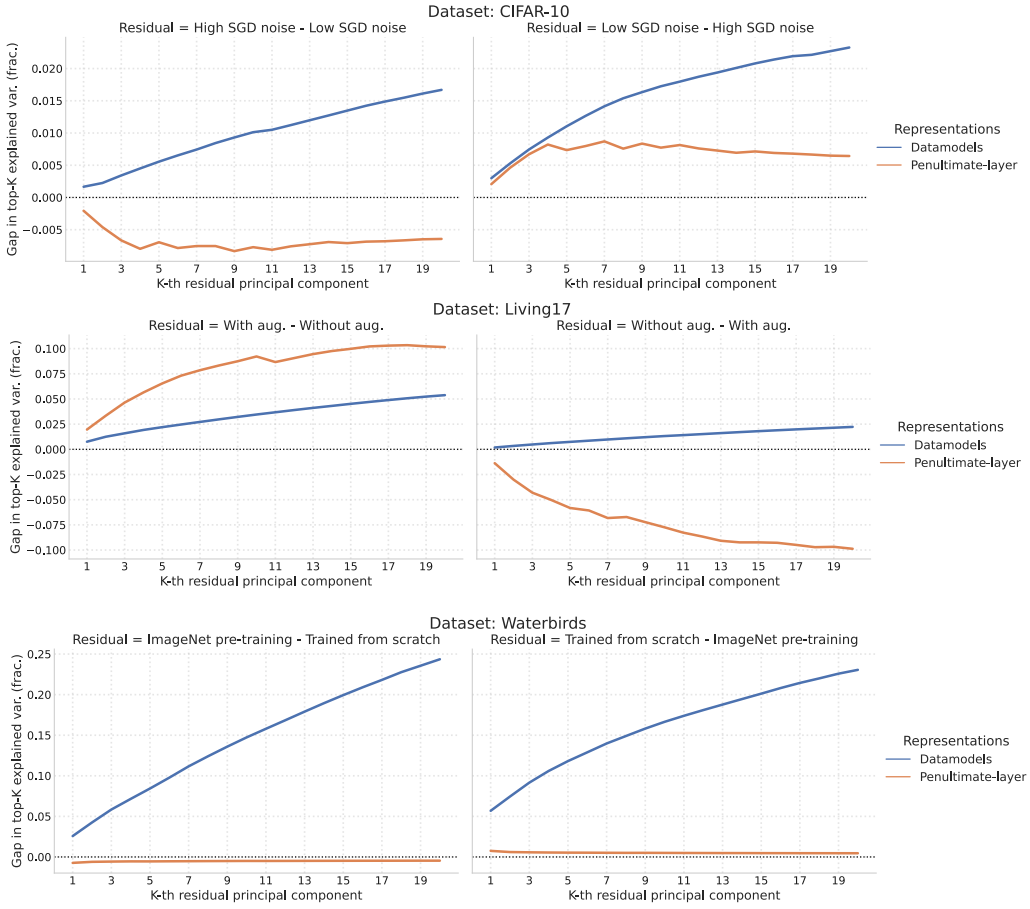


Figure 27: Difference in the cumulative fraction of variance explained by the top-k training directions across (datamodel or penultimate-layer) representations corresponding to learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$; higher the better. Top-k distinguishing training directions obtained from datamodel representations have significantly higher gap in explained variance across learning algorithms (e.g., CIFAR-10, WATERBIRDS) and (b) directions obtained from penultimate-layer representations can have close to zero (e.g., WATERBIRDS) or negative gap (e.g., CIFAR-10, LIVING17) in explained variance across learning algorithms.

G.4. Effect of prediction-level differences on distinguishing training directions

In this section, evaluate whether differences between algorithms at the model prediction level have a significant effect on the distinguishing training directions surfaced using our framework. For context, we note that there are no existing methods that analyze prediction-level differences for algorithm comparisons.

We design an experiment to show that example-level differences in predictions of models trained with different algorithms are not necessary to identify subpopulations analysed in our case studies. The first step of this experiment is to re-run our framework (a) on all test examples and (b) only on test examples on which models trained with different algorithm have the same prediction "mode" (taken over multiple runs). In the second step, we directly compare the alignment between distinguishing training directions before and after controlling for prediction-level differences.

Our results in Table 1 show that for each case study, our framework identifies similar training directions (i.e., high cosine similarity) even after removing test examples on which model predictions differ on average. This experiment shows that our framework can identify fine-grained differences between learning algorithms that persist even after controlling for prediction-level disagreement across models trained with different algorithms.

Dataset / Case study	Direction	(Absolute) Cosine Similarity
Living17 / Data augmentation	**A** (Spider web)	0.999
	B (Polka dots)	0.998
Waterbirds / ImageNet pre-training	**A** (Yellow color)	0.977
	B (Human face)	0.740
CIFAR-10 / SGD hyperparameters	**A** (Black-white texture)	0.998
	B (Rectangular shape)	0.999

Table 3: Distinguishing training directions obtained before and after filtering out high-disagreement test examples (a) exhibit high cosine similarity and (b) surface subpopulations of images that share the same distinguishing feature.

G.5. Top-k subpopulations surfaced by principal components of residual datamodels

Recall that our framework identifies distinguishing subpopulations via principal components (PCs) of residual datamodels. Specifically, these subpopulations correspond to test examples whose residual datamodel representations have the most positive (top-k) and most negative (bottom-k) projection onto a given PC. Here, we show that the top-k and bottom-k subpopulations corresponding to the top few PCs of residual datamodels considered in Section 4 surface test examples with qualitatively similar properties.

Figure 28: Top five PC subpopulations of LIVING17 residual datamodel $\theta^{(1\setminus 2)}$, where learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ correspond to training models with and without standard data augmentation respectively. Our case study in Section 4.1 analyzes PC #2 (direction **A**) and PC #5 (direction **B**).

H. Additional related work

In the main paper, we focused our discussion of related work to those closely related to the problem of algorithm comparison. Here, we give discuss additional related work on representation-based comparisons, interpretability and debugging of model biases.

Representation-based comparison methods. Our approach to algorithm comparison differs from representation-based comparison methods (Raghu et al., 2017; Morcos et al., 2018a; Kornblith et al., 2019; Bansal et al., 2021; Csiszarik et al., 2021; Chen et al., 2021; Cui et al., 2022) in both objective and implementation:

- *Learning algorithms rather than fixed models*: Rather than focusing on a single fixed model, our goal in this paper is to compare the classes of models that result from a given learning algorithm. In particular, we aim to find only differences that arise from algorithmic design choices, and not those that arise from the (sometimes significant) variability in training across random seeds (Zhong et al., 2021). Furthermore, since models exhibit significant variability in their predictions when varying only the random seeds (Zhong et al., 2021), our framework ensures that the differences we pinpoint are only those arising from the choice of the learning algorithm (and are not due to just randomness in training).
- *Feature-based rather than similarity-based*: Methods such as CCA and CKA focus on outputting a single score that reflects the overall similarity between two models. On the other hand, the goal of our framework is to find fine-grained differences in model behavior. Still, in Appendix D.1 we show that we can also use our method for more global comparisons, for instance by computing the average cosine similarity of the datamodel vectors.
- *Model-agnostic*: Our framework is agnostic to type of model used and thus allows one to easily compare models across learning algorithms—our method extends even to learning algorithms that do not have explicit representations (e.g., decision trees and kernel methods).

Interpretability, explainability, and debugging. Our method hinges on the interpretability of the extracted subpopulation. A long line of prior work propose different interpretability and explainability methods for models.

Local explanation methods include saliency maps (Simonyan et al., 2013; Dabkowski & Gal, 2017; Adebayo et al., 2018), surrogate models such as LIME (Ribeiro et al., 2016), and Shapley values (Lundberg & Lee, 2017). Our method is similar to per-example based interpretability methods such as influence functions (Koh & Liang, 2017) in that our interpretation is based on data; however, our analysis differs from these priors methods in that it looks at entire subpopulations of inputs.

Global interpretability and debugging methods often leverage the rich latent space of neural networks in order to identify meaningful subpopulations or biases more automatically. Concept activation vectors and its variants (Kim et al., 2018; Abid et al., 2022; Ghorbani et al., 2019) help decompose model predictions into a set of concepts. Other recent works (Eyuboglu et al., 2022; Jain et al., 2022) leverage the recent cross-model representations along with simple models—mixture models and SVMs, respectively—to identify coherent subpopulations or slices. Other methods (Wong et al., 2021; Singla & Feizi, 2021) analyze the neurons of the penultimate layer of (adversarially robust) models to identify spurious features. Our framework can be viewed as leveraging a different embedding space, that of datamodel representations, to analyze model predictions.

Robustness to specific biases. In applying our framework across the three case studies, we identify a number of both known and unknown biases. A large body of previous work aims at finding and debugging these biases: Priors works investigate specific biases such as the role of texture (Geirhos et al., 2019) or backgrounds (Xiao et al., 2020) by constructing new datasets. Leclerc et al. (2021) automate many of these studies in the context of vision models with a render-based framework. Relatedly, Ruiz et al. (2022) develop a simulation-based testing environment to compare predictions of vision transformers and convolutional networks under naturalistic scene variations such as object pose and camera viewpoint. In contrast to our approach, these works rely on having control over data generation and having candidate biases ahead of time.

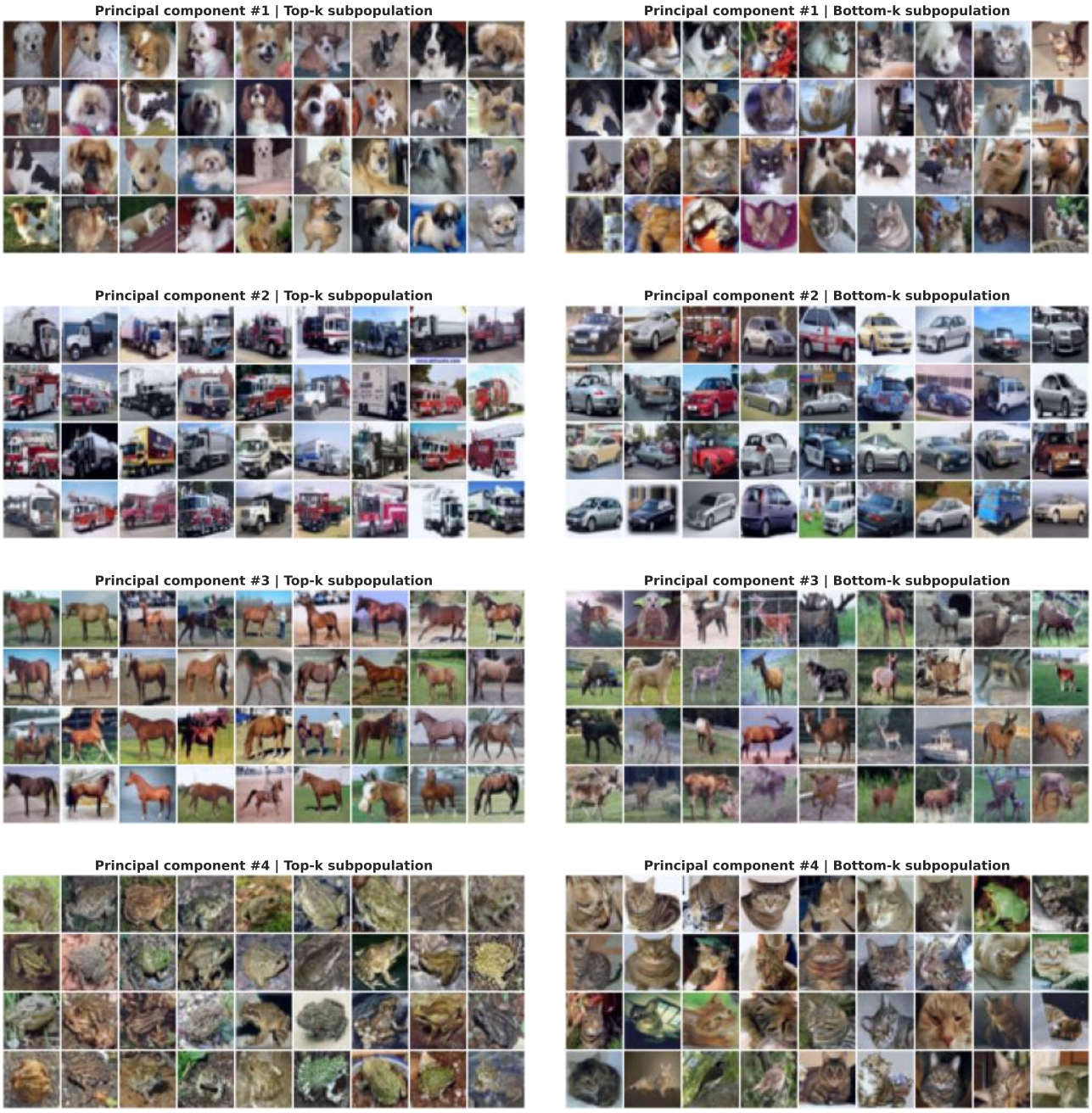


Figure 29: Top four PC subpopulations of CIFAR-10 residual datamodel $\theta^{(2\setminus 1)}$, where learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ correspond to training models with high and low SGD noise respectively. Our case study in Appendix C.1 analyzes PC #1 (direction **A**) and PC #2 (direction **B**).

Figure 30: Top four PC subpopulations of WATERBIRDS residual datamodel $\theta^{(1\setminus 2)}$, where learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ correspond to training models with and without ImageNet pre-training respectively. Our case study in Section 4.1 analyzes PC #3 (direction **A**).

Figure 31: Top four PC subpopulations of WATERBIRDS residual datamodel $\theta^{(2\setminus 1)}$, where learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ correspond to training models with and without ImageNet pre-training respectively. Our case study in Section 4.1 analyzes PC #3 (direction **B**).