A Unified Framework for Comparing Learning Algorithms

Harshay Shah* harshay@mit.edu MIT
Sung Min Park* sp765@mit.edu MIT
Andrew Ilyas* ailyas@mit.edu MIT
Aleksander Mądry madry@mit.edu MIT

Abstract

Understanding model biases is crucial to understanding how models will perform out-of-distribution (OOD). These biases often stem from particular design choices (e.g., architecture or data augmentation). We propose a framework for *(learning) algorithm comparisons*, wherein the goal is to find similarities and differences between models trained with two different learning algorithms. We begin by formalizing the goal of algorithm comparison as finding *distinguishing feature transformations*, input transformations that change the predictions of models trained with one learning algorithm but not the other. We then present a two-stage method for algorithm comparisons based on comparing how models use the training data, leveraging the recently proposed datamodel representations [IPE+22]. We demonstrate our framework through a case study comparing classifiers trained on the WATERBIRDS [SKH+20] dataset with/without ImageNet pre-training.

1 Introduction

How models perform out-of-distribution (OOD) is shaped in part by the biases of these models. These biases are often influenced by the particular design choices we make. For example, Hooker et al. [HCC+19] find that model compression—a common design choice prior to deployment—preserves test set accuracies but can significantly degrade OOD performance. In order to understand how design choices—which together define a *learning algorithm*—impact model biases, we thus need to be able to differentiate learning algorithms in a more fine-grained way than accuracy alone.

Motivated by this observation, we develop a unified framework for comparing learning algorithms. Our proposed framework comprises (a) a precise, quantitative definition of learning algorithm comparison; and (b) a concrete methodology for comparing any two algorithms. For (a), we frame the algorithm comparison problem as one of finding input transformations that *distinguish* the two algorithms. This goal is different and more general than quantifying model similarity [DDS21; BNB21; MRB18] or testing specific biases [HCK20]. For (b), we propose a two-stage method for comparing algorithms in terms of *how they use the training data*.

In the first stage of this method, we leverage *datamodel representations* [IPE+22] to find weighted combinations of training examples (which we call *training directions*) that have disparate impact on test-time behavior of models across learning algorithms. In the second stage, we filter the subpopulation of test examples that are most influenced by each identified training direction, then manually inspect them to infer a shared feature (e.g., all images contain a human in the background). We then tie this intuition back to our quantitative definition by designing a distinguishing feature transformation based on the shared feature (e.g., adding a human to the background).

We demonstrate our framework through a case study comparing classifiers first pre-trained on ImageNet [DDS+09; RDS+15] then fine-tuned on WATERBIRDS [SKH+20] with classifiers trained from scratch on WATERBIRDS. Our results demonstrate that ImageNet pre-training reduces dependence of models on some spurious correlations (e.g., yellow color $\rightarrow$ landbird) but also *introduces* new ones (e.g., human face in the background $\rightarrow$ landbird).

Workshop on Distribution Shifts, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

2 Comparing learning algorithms

In this section, we describe our (learning) algorithm comparison framework. In Section 2.1, we formalize algorithm comparison as the task of identifying *distinguishing transformations*. These are functions that—when applied to test examples—significantly and consistently change the predictions of one model class but not the other. In Section 2.2, we describe our method for identifying distinguishing feature transformations by comparing how each model class uses the training data.

2.1 Formalizing algorithm comparisons via distinguishing transformations

The goal of algorithm comparison is to understand the ways in which two learning algorithms (trained on the same data distribution) differ in the models they yield. More specifically, we are interested in comparing the *model classes* induced by the two learning algorithms:

Definition 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 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 $\mathcal{D}$.*

We adopt the perspective that model classes differ insofar as they use different features to make predictions. To make this precise, we introduce the notion of *distinguishing feature transformations*:

Definition 2 (Distinguishing feature transformation)**.** *Let $\mathcal{A}_1, \mathcal{A}_2$ denote learning algorithms, S a dataset of input-label pairs, and $\mathcal{L}$ a loss function.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 feature transformation of M_1 with respect to M_2 is a function $F : \mathcal{X} \to \mathcal{X}$ such that for some label $y_c \in \mathcal{Y}$,*

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

where $L_i(x, y) = \mathcal{L}(M_i(x), y)$, and the expectations above are taken over both inputs x and randomness in the learning algorithm.

Intuitively, a distinguishing feature transformation is just a function F that, when applied to test data points, significantly changes the predictions of one model class—but not the other—in a consistent way. Definition 2 also immediately suggests a way to *evaluate* the effectiveness of a distinguishing feature transformation. That is, given a hypothesis about how two algorithms differ (e.g., that models trained with $\mathcal{A}_1$ are more sensitive to texture than those trained with $\mathcal{A}_2$), one can design a corresponding transformation F (e.g., applying style transfer, as in [GRM+19]), and directly measure its relative effect on the two model classes.

Since our goal is to qualitatively understand the differences in the biases of the model classes, we look for distinguishing feature transformations that are *informative*, i.e., they (a) capture a feature that naturally arises in the data distribution and (b) are semantically meaningful.

2.2 Identifying distinguishing feature transformations

We now describe our two-stage method for comparing learning algorithms based on *how they use the training data*. We first identify *training directions*, or weighted combinations of training examples, that impact test performance of models trained with one learning algorithm but not the other. Then, we use a human-in-the-loop approach to extract semantically meaningful features and corresponding transformations from these directions.

Stage I: An algorithm for finding distinguishing training directions. First, we find weighted combinations of training examples that influence the predictions of models trained with one learning algorithm but not the other. Our algorithm leverages *datamodel representations* [IPE+22] and comprises the following three steps:

1. Compute datamodels for each algorithm. Given training set S and learning algorithm $\mathcal{A}$, a datamodel (representation) for test example x_i is a vector $\theta_i \in \mathbb{R}^{|S|}$, where θ_{ij} measures the extent

to which models trained with $\mathcal{A}$ depend [2] on the j-th training example to correctly classify example x_i. We compute two sets of datamodels—$\theta^{(1)}$ and $\theta^{(2)}$—corresponding to model classes induced by learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively.

2. Compute residual datamodels. Next, we compute a *residual datamodel* for each test example x_i, which is the projection of the datamodel $\theta_i^{(1)}$ onto the null space of datamodel $\theta_i^{(2)}$, i.e.,

$$\theta_i^{(1\backslash 2)} = \hat{\theta}_i^{(1)} - \langle \hat{\theta}_i^{(1)}, \hat{\theta}_i^{(2)} \rangle \, \hat{\theta}_i^{(2)}$$

where $\hat{\theta}_i = \theta_i / \|\theta_i\|_2$ denotes the normalized version of datamodel θ_i. Intuitively, the residual datamodels of algorithm $\mathcal{A}_1$ with respect to $\mathcal{A}_2$ correspond to the training directions that influence $\mathcal{A}_1$ after "projecting away" the component that also influences $\mathcal{A}_2$.

3. Run principal component analysis. Finally, we use principal component analysis (PCA) to find the highest-variance directions in the space of residual datamodels. That is, we run

$$\{e_1^{(1\backslash 2)}, \ldots, e_\ell^{(1\backslash 2)}\} \leftarrow \ell\text{-PCA}(\{\theta_1^{(1\backslash 2)}, \ldots, \theta_{|T|}^{(1\backslash 2)}\})$$

to find the top ℓ principal components of the residual datamodels. Intuitively (deferring formal analysis to Appendix B), we expect the returned principal components to be the most distinguishing training directions across the test set.

Stage II: Human-in-the-loop analysis. With these distinguishing directions in hand, we use a human-in-the-loop analysis to identify *informative* distinguishing features transformations in three steps. First, given a principal component, we inspect the test examples whose residual datamodels are most aligned with that component. We view these examples as representing *subpopulations* that depend most heavily on the training direction. With this subpopulation, we then infer a *distinguishing feature* by visual inspectionand if needed, additional analysis. Finally, we design a distinguishing feature transformation to counterfactually verify the effect of the inferred feature on model behavior.

We illustrate our algorithm visually in the top half of Figure 1.

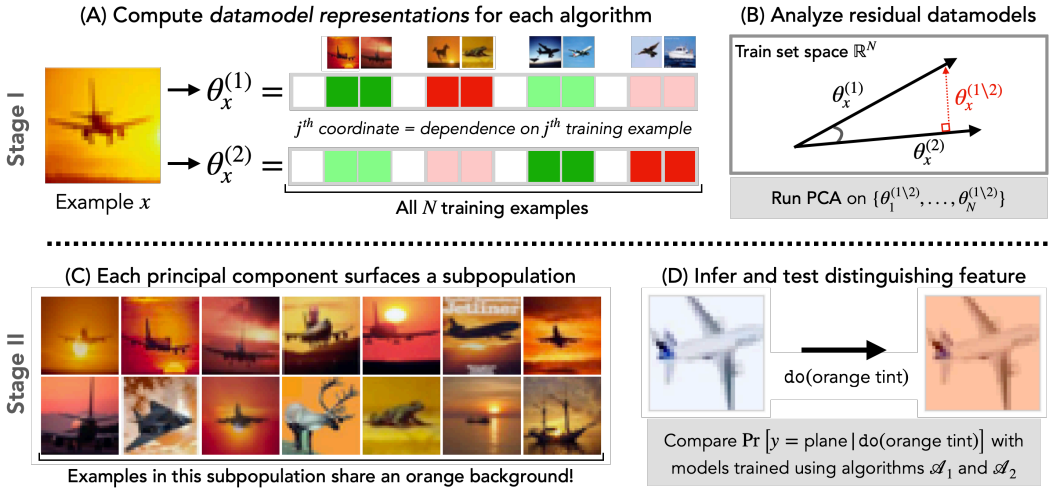


Figure 1: A visual summary of our two-stage approach to algorithm comparison. In the first stage (top row) we use examples' datamodel representations [IPE+22] to find so-called *distinguishing training directions*—weighted combinations of training examples that impact the two algorithms disparately across the test set. In the second stage (bottom row) we surface *subpopulations* that rely on the identified directions, and use a human-in-the-loop to go from the identified distinguishing training direction to a testable feature transformation.

[2] For readers familiar with influence functions [KL17; HRR+11], an intuitive (but not quite accurate) way to interpret datamodel weight θ_{ij} is as the influence of the j-th training example on test example x_i. In Section B.1, we discuss additional properties of datamodel representations that make them particularly well-suited for algorithm comparison.

3 Applying the algorithm comparison framework

We now demonstrate our comparison framework through a case study comparing models trained with and without pre-training.

Setup. We study the effect of ImageNet pre-training [DDS+09; KSL19] in the context of classifiers trained on the WATERBIRDS dataset[3] [SKH+20]. ImageNet pre-training significantly improves the "worst group" accuracy on WATERBIRDS images where the background conflicts with the bird—but how does it impact the fine-grained biases of the models? To study this, we compare two classes of ResNet-50 models trained with the exact same settings modulo the use of ImageNet pre-training. That is, we consider the following two learning algorithms, with additional details in Appendix C.2.

- **Algorithm** $\mathcal{A}_1$: Pre-training on ImageNet, followed by full-network finetuning on WATERBIRDS data. Models trained this way attain 89.1% average accuracy and 63.6% worst-group accuracy.
- **Algorithm** $\mathcal{A}_2$: Training from scratch on WATERBIRDS data. Corresponding models attain 63.9% average accuracy and 5.7% worst-group accuracy on the test set.

Identifying distinguishing features. We compare algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ using our method from Section 2. The first stage of this method finds the top l *distinguishing training directions* $\{v_1, ..., v_l\} \in \mathbb{R}^{|S|}$. As we will show (in Appendix B), for a given training direction v, the fraction of variance that v explains [4] in the datamodel representations $\{\theta_x^{(i)}\}$ captures the importance of the corresponding combination of training examples to model predictions for algorithm $\mathcal{A}_i$. Thus, we would hope for training directions that distinguish $\mathcal{A}_1$ from $\mathcal{A}_2$ to explain a high (resp., low) amount of the variance in datamodel representations of algorithm $\mathcal{A}_1$ (resp., $\mathcal{A}_2$).

Figure 2 displays the identified distinguishing training directions. On the left, we can see that the training directions distinguishing $\mathcal{A}_1$ from $\mathcal{A}_2$ (in green) indeed explain a significant amount of variance in the datamodels of $\mathcal{A}_1$ but not in those of $\mathcal{A}_2$. Visualizing the subpopulations corresponding to two of the distinguishing directions (Figure 2 right) suggests the following distinguishing features:

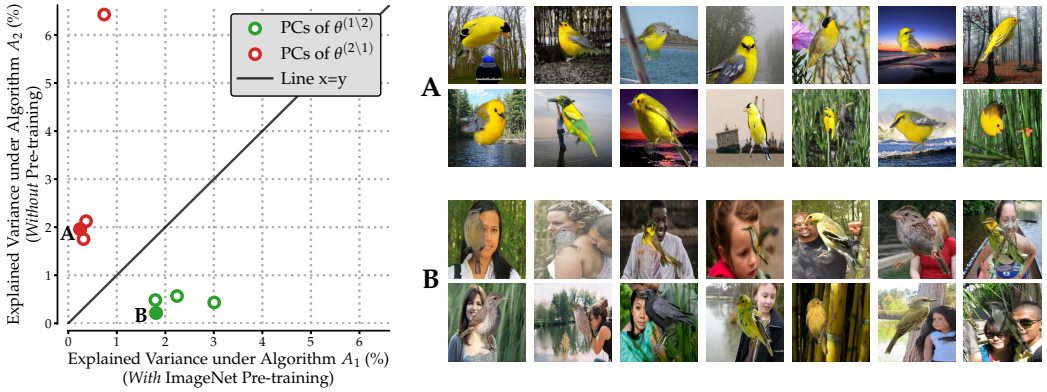


Figure 2: **Comparing Waterbirds models trained with and without ImageNet pre-training. (Left)** Each green (resp., red) point is a *training direction* (i.e., a vector $v \in \mathbb{R}^{|S|}$ representing a weighted combination of training examples) that distinguishes $\mathcal{A}_1$ from $\mathcal{A}_2$ (resp., $\mathcal{A}_2$ from $\mathcal{A}_1$) as identified by the first stage of our framework. The x and y coordinates of each point represent the importance of the training direction to models trained with $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. **(Right)** Test examples most impacted by the distinguishing training directions annotated **A** and **B**. Direction **A** seems to correspond to yellow birds and direction **B** to human faces in the background.

[3] Waterbirds is an image dataset of bird foregrounds pasted on landscape backgrounds, where the task is to classify between "land birds" and "water birds" based on only the foreground, despite a strong spurious correlation between background and foreground in the training set.

[4] The *fraction of explained variance* of a given vector $v \in \mathbb{R}^d$ in a set of vectors $\{\theta_x \in \mathbb{R}^d\}$ is the empirical variance of $v^\top \theta_x$ divided by the total amount of variance in $\{\theta_x\}$ (i.e., trace(Cov$[\theta_x]$)). In other words, this measures what fraction of the total variation in $\{\theta_x\}$ is along the direction v.

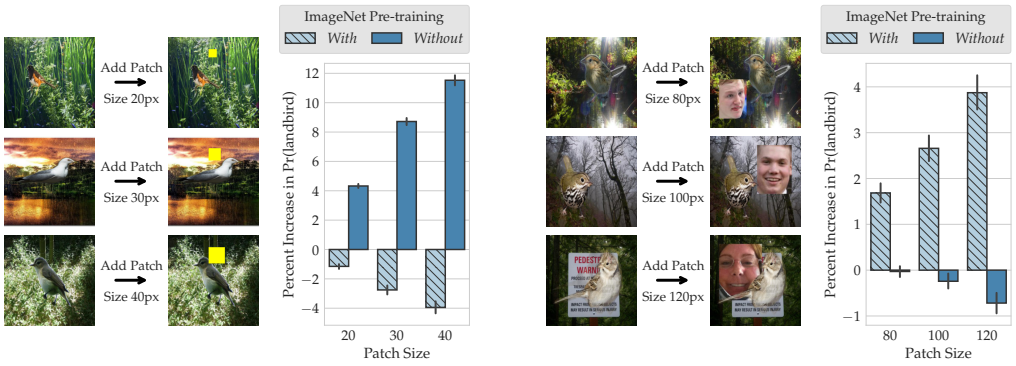


Figure 3: **Effect of ImageNet pre-training on WATERBIRDS classification**. We use our framework to identify two distinguishing features: **(Left)** Adding a yellow patch to images makes models trained without (with) pre-training, on average, 9% more (2% less) confident in predicting "landbird." **(Right)** Adding faces to image backgrounds makes models trained with (without) pre-training, on average, 3% (0%) more confident in predicting "landbird." In both cases, increasing the intensity of feature transformations widens the gap in treatment effect between the two model classes.

Yellow color: Direction **A** surfaces a subpopulation of test images that contain yellow "landbird" images. This leads us to hypothesize that models trained from scratch spuriously relies on the color yellow to predict the class "landbird," whereas ImageNet-pretrained models do not. Additional analysis supports this hypothesis, as training images that contain other yellow objects strongly influence the predictions of models trained from scratch on this subpopulation (Appendix D). To test this hypothesis, we design a feature transformation that adds a yellow square patch to images (Figure 3).

Human face: Direction **B** surfaces a subpopulation of "landbird" that have *human faces* in the background. This suggests that ImageNet pre-training introduces a spurious dependence on human faces to predict the label "landbird." Additional analysis supports this hypothesis, as training images containing human face(s) strongly influence the predictions of ImageNet-pretrained models on this subpopulation (see Appendix D). To test this hypothesis, we design a feature transformation that inserts patches of human faces to WATERBIRDS image backgrounds (Figure 3).

Findings. In Figure 3, we compare the effect of the above feature transformations on models trained with and without ImageNet pre-training. The results confirm both of our hypotheses. Adding a yellow square patch with varying size (20/30/40 px) to test images increased $P(\text{"landbird"})$ by 4%/9%/12% for models trained from scratch but *decreased* $P(\text{"landbird"})$ for models pre-trained on ImageNet. Similarly, adding a human face patch to image backgrounds increased $P(\text{"landbird"})$ by 2%/3%/4% for pre-trained models, but did not significantly affect models trained from scratch. Furthermore, increasing the intensity (i.e., patch size) of these feature transformations further widens the gap in sensitivity between the two model classes. These differences verify that the feature transformations we constructed can distinguish the two learning algorithms as according to Definition 2. We defer additional analysis and details on feature transformations to Appendix E.

Connections to prior work. This case study pinpoints how pre-training can alter the importance of different spurious correlations. In particular, our results show that ImageNet pre-training reduces dependence on some spurious correlations (e.g., yellow color $\rightarrow$ landbird) but also *introduces* new ones (e.g., human face $\rightarrow$ landbird). Our findings thus shed light on two seemingly contradictory phenomena: pre-training can *simultaneously* improve robustness to spurious features [GML22; TLG+20] in target data as well as transfer new spurious correlations [SJI+22; NSZ20] from the pre-training dataset.

4 Conclusion

We introduce a unified framework for fine-grained comparisons of any two learning algorithms. Specifically, our framework compares models trained using two different algorithms in terms of how the models *rely on training data* to make predictions. In our case study, we applied our framework to pinpoint how pre-training can amplify or suppress specific spurious correlations.

References

[AGM+18] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. "Sanity checks for saliency maps". In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[BNB21] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. "Revisiting Model Stitching to Compare Neural Representations". In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[BWW+21] Manel Baradad Jurjo, Jonas Wulff, Tongzhou Wang, Phillip Isola, and Antonio Torralba. "Learning to see by looking at noise". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 2556–2569.

[CKM+21] Adrian Csiszarik, Peter Korosi-Szabo, Akos Matszangosz, Gergely Papp, and Daniel Varga. "Similarity and Matching of Neural Network Representations". In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[CKM+22] Tianyu Cui, Yogesh Kumar, Pekka Marttinen, and Samuel Kaski. "Deconfounded Representation Similarity for Comparison of Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[CLH+21] Zuohui Chen, Yao Lu, JinXuan Hu, Wen Yang, Qi Xuan, Zhen Wang, and Ziaoniu Yang. "Revisit Similarity of Neural Network Representations From Graph Perspective". In: *arXiv preprint arXiv:2111.11165* (2021).

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *Computer Vision and Pattern Recognition (CVPR)*. 2009.

[DDS21] Frances Ding, Jean-Stanislas Denain, and Jacob Steinhardt. "Grounding Representation Similarity with Statistical Testing". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[DS22] Jean-Stanislas Denain and Jacob Steinhardt. "Auditing Visualizations: Transparency Methods Struggle to Detect Anomalous Behavior". In: *arXiv preprint arXiv:2206.13498* (2022).

[FZ20] Vitaly Feldman and Chiyuan Zhang. "What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 2881–2891.

[GML22] Soumya Suvra Ghosal, Yifei Ming, and Yixuan Li. "Are Vision Transformers Robust to Spurious Correlations?" In: *arXiv preprint arXiv:2203.09125* (2022).

[GRM+19] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness." In: *International Conference on Learning Representations (ICLR)*. 2019.

[HCC+19] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. "What Do Compressed Deep Neural Networks Forget? 1911.05248". In: *arXiv preprint arXiv:1911.05248*. 2019.

[HCK20] Katherine Hermann, Ting Chen, and Simon Kornblith. "The Origins and Prevalence of Texture Bias in Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. 2020.

[HEK+18] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. "A benchmark for interpretability methods in deep neural networks". In: *arXiv preprint arXiv:1806.10758* (2018).

[HRR+11] Frank R Hampel, Elvezio M Ronchetti, Peter J Rousseeuw, and Werner A Stahel. *Robust statistics: the approach based on influence functions*. Vol. 196. John Wiley & Sons, 2011.

[IPE+22] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. "Datamodels: Predicting Predictions from Training Data". In: *International Conference on Machine Learning (ICML)*. 2022.

[KL17] Pang Wei Koh and Percy Liang. "Understanding Black-box Predictions via Influence Functions". In: *International Conference on Machine Learning*. 2017.

[KNL+19] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. "Similarity of Neural Network Representations Revisited". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019.

[KSL19] Simon Kornblith, Jonathon Shlens, and Quoc V Le. "Do better imagenet models transfer better?" In: *computer vision and pattern recognition (CVPR)*. 2019.

[LIE+22] Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. *ffcv*. https://github.com/libffcv/ffcv/. 2022.

[LL17] Scott Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[LYC+15] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. "Convergent Learning: Do different neural networks learn the same representations?" In: *Proceedings of the 1st International Workshop on Feature Extraction: Modern Questions and Challenges at NIPS 2015*. 2015.

[MBG+22] Kristof Meding, Luca M. Schulze Buschoff, Robert Geirhos, and Felix A. Wichmann. "Trivial or Impossible — dichotomous data difficulty masks model differences (on ImageNet and beyond)". In: *International Conference on Learning Representations (ICLR)*. 2022.

[MMS+19] Horia Mania, John Miller, Ludwig Schmidt, Moritz Hardt, and Benjamin Recht. "Model similarity mitigates test set overuse". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019, pp. 9993–10002.

[MRB18] Ari Morcos, Maithra Raghu, and Samy Bengio. "Insights on representational similarity in neural networks with canonical correlation". In: *Advances in Neural Information Processing Systems* 31 (2018).

[NRK21] Thao Nguyen, Maithra Raghu, and Simon Kornblith. "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.

[NSZ20] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. "What is being transferred in transfer learning?" In: *Advances in neural information processing systems* 33 (2020), pp. 512–523.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)*. 2015.

[RGY+17] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. "SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability". In: *Advances in Neural Information Processing Systems*. 2017.

[RUK+21] Maithra Raghu, Thomas Unterthiner, Simon Kornblith, Chiyuan Zhang, and Alexey Dosovitskiy. "Do Vision Transformers See Like Convolutional Neural Networks?" In: *Neural Information Processing Systems (NeurIPS)*. 2021.

[SJI+22] Hadi Salman, Saachi Jain, Andrew Ilyas, Logan Engstrom, Eric Wong, and Aleksander Madry. "When does Bias Transfer in Transfer Learning?" In: *arXiv preprint arXiv:2207.02842*. 2022.

[SJN21] Harshay Shah, Prateek Jain, and Praneeth Netrapalli. "Do Input Gradients Highlight Discriminative Features?" In: *Advances in Neural Information Processing Systems* 34 (2021).

[SKH+20] Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. "Distributionally Robust Neural Networks for Group Shifts: On the Importance of Regularization for Worst-Case Generalization". In: *International Conference on Learning Representations*. 2020.

[TLG+20] Lifu Tu, Garima Lalwani, Spandana Gella, and He He. "An empirical study on robustness to spurious correlations using pre-trained language models". In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 621–633.

[WBS+20] John Wu, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. "Similarity Analysis of Contextual Word Representation Models". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020.

[WBW+11] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. "The caltech-ucsd birds-200-2011 dataset". In: (2011).

[WWZ+22] Junpeng Wang, Liang Wang, Yan Zheng, Chin-Chia Michael Yeh, and Shubham Jain andWei Zhang. "Learning-From-Disagreement: A Model Comparison and Visual Analytics Framework". In: *arXiv preprint arXiv:2201.07849* (2022).

[YYF+22] Kaiyu Yang, Jacqueline H Yau, Li Fei-Fei, Jia Deng, and Olga Russakovsky. "A study of face obfuscation in imagenet". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 25313–25330.

[ZGK+21] Ruiqi Zhong, Dhruba Ghosh, Dan Klein, and Jacob Steinhardt. "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.

[ZLK+17] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. "Places: A 10 million image database for scene recognition". In: *IEEE transactions on pattern analysis and machine intelligence*. 2017.

Appendices

A Related work **10**

B Algorithm analysis **12**
B.1 A primer on datamodel representations 12
B.2 Residual datamodels 13
B.3 Finding global trends with PCA 13

C Experimental Setup **14**
C.1 Dataset 14
C.2 Models, learning algorithms, and hyperparameters 14
C.3 Datamodels 14
C.4 Feature transformations 15
C.5 Training infrastructure 15

D Additional human-in-the-loop analysis **17**
D.1 Tools for Inferring distinguishing features from PCA subpopulations 17
D.2 Case study: ImageNet pre-training 18

E Additional evaluation of distinguishing feature transformations **20**

F Miscellaneous results **21**
F.1 Aggregate metric for algorithm comparison 21
F.2 Explained variance of residual datamodel principal components 22
F.3 Subpopulations surfaced by principal components of residual datamodels 23

A Related work

We place our work into context with existing approaches to model and algorithm comparison in machine learning.

In particular, we 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. For simple models (e.g., sparse linear), we can directly compare models in parameter space. However, distances between more complex models such as deep neural networks are much less meaningful. To this end, a long line of work has sought to design methods for characterizing the differences between models:

Representation-based comparison. A popular approach (particularly in the context of deep learning) is to compare models using their internal *representations*. Since the coordinates of these representations do not have a consistent interpretation, representation-based model comparison typically studies the degree to which different models' representations can be aligned. Methods based this approach include canonical correlation analysis (CCA) and variants [RGY+17; MRB18; CKM+22], centered kernel alignment (CKA) [KNL+19], graph-based methods [LYC+15; CLH+21], and model stitching [CKM+21; BNB21]. Prior works have used these methods to compare wide and deep neural networks [NRK21]; vision transformers and convolutional networks [RUK+21]; pre-trained and trained-from-scratch models [NSZ20]; and different language models [WBS+20]. Though they are often useful, prior work shows that representation-based similarity measures are not always statistically reliable for testing functional differences in models [DDS21]. Our approach to algorithm comparison differs from these methods in both objective and implementation:

- *Learning algorithms rather than fixed models*: Rather than focusing on a single fixed model, our objective in this paper is to compare the class 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 [ZGK+21].
- *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 F.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).

Example-level comparisons. An alternative method for comparing models is to compare their predictions directly. For example, Zhong et al. [ZGK+21] compare predictions of small and large language models (on a per-example level) to find that larger models are not uniformly better across example. Similarly, Mania et al. [MMS+19] study the *agreement* between models, i.e., how often they output the same prediction on a per-example level. In another vein, Meding et al. [MBG+22] show that after removing impossible or trivial examples from test sets, different models exhibit more variations in their predictions. Our framework also studies instance-level predictions, but ultimately connects the results back to human-interpretable distinguishing features.

Comparing feature attributions. Finally, 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., using SHAP [LL17]) and compare them across models [WWZ+22]. In cases where we do not have access to high-level features, we can use instance-level explanation methods such as saliency maps to highlight different parts of the input, but these methods generally do not help at distinguishing models [DS22]. Furthermore, multiple evaluation metrics [AGM+18; HEK+18; SJN21] indicate that common instance-specific feature attribution methods can fail at accurately highlighting features learned by the model.

Prior work on pre-training. Our case study pinpoints how pre-training can alter the importance of different spurious correlations. In particular, our results show that ImageNet pre-training reduces dependence on some spurious correlations (e.g., yellow color $\rightarrow$ landbird) but also *introduces* new ones (e.g., human face $\rightarrow$ landbird). Our findings thus shed light on two seemingly contradictory phenomena: pre-training can *simultaneously* improve robustness to spurious features [GML22; TLG+20] in target data as well as transfer new spurious correlations [SJI+22; NSZ20] from the pre-training dataset.

B Algorithm analysis

In the main paper, we applied our comparison framework to identify feature transformations that distinguished three pairs of learning algorithms. Here, we describe in more detail the algorithmic stage of that framework, i.e., the stage whose purpose is to find *distinguishing training directions*. To this end, we walk through each of the three steps of the algorithm presented in Section 2.2 and provide intuition for how they identify distinguishing training directions.

B.1 A primer on datamodel representations

The first step in our method is to compute datamodel vectors $\theta_j^{(i)} \in \mathbb{R}^{|S|}$, one for each test input x_j. Each datamodel vector encodes the importance of individual training examples S to model's loss at input x_j when trained with learning algorithm $\mathcal{A}_i$. More specifically, each vector corresponds to the solution to a specific regression problem—these regression problems (explained below) form the basis of our analysis.

Setting up the regression problem. Let us fix a single learning algorithm $\mathcal{A}$ (being $\mathcal{A}_1$ or $\mathcal{A}_2$). For a given training set $S = \{x_1, \ldots, x_d\}$, a test input x, and subset $S' \subset S$ of the training set we define the *model output function* as:

$$f(x, S') = \text{ the loss after training a model on } S' \text{ and evaluating on } x.$$

For example, $f(x, S')$ can encode training a deep neural network on the subset S', then computing the network's *correct-class margin* on the input x. For a fixed x, the corresponding regression problem is to predict the model output $f(x, S')$ given a subset S'.

Datamodels. Ilyas et al. [IPE+22] show that—for deep neural networks trained on standard image classification tasks—we can solve the regression problem above with a simple *linear* predictor. More specifically, they showed that

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

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

$$(\mathbf{1}_{S'})_i = \begin{cases} 1 & \text{if } x_i \in S' \\ 0 & \text{otherwise.} \end{cases}$$

In this way, datamodels consistitute linear approximations of model output functions.

Datamodels as a representation space. While each datamodel is specific to an individual test input, we can treat a collection of datamodels as *embeddings or representations* of test inputs $\{x_j\}_j$ into a common $|S|$-dimensional space. By comparing the representations, we can analyze the structure of the data—as used by the specific learning algorithm under study. Furthermore, these representations have a number of properties that make them useful for algorithm comparisons:

(a) **Consistent basis:** A datamodel representation for a fixed train set S always has the same basis: coordinate i corresponds to the importance of the i-th training example. This consistency makes datamodels a convenient medium for algorithm comparisons, as representations are *automatically aligned* across different learning algorithms, and even across models that lack explicit representations (e.g., decision trees).

(b) **Predictiveness:** Datamodel vectors are *causally predictive* of model behavior. That is, as Ilyas et al. [IPE+22] show, we can use them to predict the counterfactual impact of removing or adding different training examples on model output for a given test example. As a result, any trends we find across the datamodel representations come with a precise quantitative interpretation in terms of model outputs (to which we will come back to later in thise section).

(c) **Density:** Datamodel representations also have a *high effective dimensionality*: that is, one needs thousands of components to explain significant fraction of variance, for instance, on CIFAR-10 [IPE+22].[5] This suggests that datamodel representations encode fine-grained information about

[5] This is in stark contrast to "standard" representations derived from the penultimate layer of a trained model, which tend to have effective dimensions that are much lower, typically equal to number of classes minus one.

how each learning algorithm uses the training data, making them useful for uncovering subtle differences in model behavior.

B.2 Residual datamodels

In Step 2 of our algorithm, using the two sets of datamodels $\{\theta^{(1)}\}$ and $\{\theta^{(2)}\}$, we compute the residual datamodel vectors:

$$\theta_i^{(1\setminus 2)} = \theta_i^{(1)} - \langle \theta_i^{(1)}, \theta_i^{(2)} \rangle \, \theta_i^{(2)},$$

(Note that this operation only makes sense because the two sets of datamodels live in the same vector space—see property (a) above.) As we demonstrate, they correspond to datamodels for a certain *residual model output* function that we define below.

Recall that our overarching goal is to find training directions that that strongly influence models trained with learning algorithm $\mathcal{A}_1$ but not $\mathcal{A}_2$ (or vice-versa) when classifying x. More specifically, we want to find training directions u that strongly influences $f^{(1)}(x, S)$, while ignoring directions that also influence $f^{(2)}(x, S)$. In other words, what we care about is the "residual" of $f^{(1)}$ after removing the part that is *correlated*[6] with $f^{(2)}$. To capture this, consider the *residual model output* function of $\mathcal{A}_1$ relative to $\mathcal{A}_2$:

$$f^{(1\setminus 2)}(x, S) \coloneqq f^{(1)}(x, S) - \rho_{f^{(1)} f^{(2)}} \cdot f^{(2)}(x, S)$$

where $\rho_{f^{(1)} f^{(2)}}$ is the correlation between two model output functions (across varying S). It turns out that the datamodel for the residual output function is given precisely by the *residual datamodel* defined in Step 2 of our algorithm: the projection of one (normalized) datamodel representation into the nullspace of the other representation. That is, residual datamodels correspond to a linear approximation of the residual model output:

$$\mathbb{E}[f^{(1\setminus 2)}(x, S)] \approx \theta^{(1\setminus 2)} \cdot \mathbf{1}_S$$

In summary, Step 2 of our procedure reduces understanding the differences in learning algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ to analyzing their residual model outputs via residual datamodels. The residual datamodels highlight directions that influence learning algorithm $\mathcal{A}_1$ after *projecting away* directions that also influence learning algorithm $\mathcal{A}_2$.

B.3 Finding global trends with PCA

The output of Step 2 of our procedure is two set of residual datamodels—one set looking at the residual of $\mathcal{A}_1$ with respect to $\mathcal{A}_2$, and vice versa. These residual datamodels capture directions that each algorithm is most sensitive to, though still on a *per-example* level. On the other hand, our goal is to find directions that each algorithm is most sensitive to on an *aggregate* level (as in Definition 2). We now show how the final step in our procedure, computing PCA on the residual datamodel representations, achieves this goal.

For a given input x with datamodel θ, and for a training direction $u \in \mathbb{R}^{|S|}$, we can estimate the example's sensitivity to u as $[(\theta \cdot u)]^2$—that is, the (estimated) effect on $f(x, S)$ of upweighting and downweighting the training samples according to u. Since our goal is to find a direction that the residual model output function $f^{(1\setminus 2)}$ is most sensitive *in aggregate*, our objective is

$$\arg \max_{u \in \mathbb{S}^{n-1}} \mathbb{E}_{x \in T}[(\theta_x^{(1\setminus 2)} \cdot u)^2].$$

The direction that maximizes this objective is exactly the dominant principal component of the set of residual datamodels! Similarly, the top k principal components correspond to directions that algorithm $\mathcal{A}_1$ is most sensitive to after accounting for $\mathcal{A}_2$.

In summary, Step 3 of our procedure finds directions that strongly influence only one learning algorithm by examining directions of highest explained variance of the residual datamodel vectors.

[6] Over the distribution of S.

C Experimental Setup

In this section, we outline the experimental setup—datasets, models, training algorithms, hyperparameters, and datamodels—used for our case study in Section 3.

C.1 Dataset

The Waterbirds dataset [SKH+20] consists of bird images taken from the CUB dataset [WBW+11] and pasted on backgrounds from the Places dataset [ZLK+17]. 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. [SKH+20] 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.

Summary statistics of the WATERBIRDS dataset are outlined in Table 1.

Table 1: Summary statistics of dataset

Dataset	Classes	Size (Train/Test)	Input Dimensions
Waterbirds	2	4,795/5,794	$3 \times 224 \times 224$

C.2 Models, learning algorithms, and hyperparameters

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. [SKH+20], 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.

C.3 Datamodels

Now, we provide additional details on datamodels which, we recall, are used in the first stage of our algorithm comparison framework (see Section 2).

Estimating linear datamodels. Recall from Appendix B 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:

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

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. Ilyas et al. [IPE+22] demonstrate that linear datamodels can accurately predict outputs of deep image classifiers.

Datamodel estimation hyperparameters. Recall that our algorithm comparison framework in Section 2 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 study, 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. [IPE+22], 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 [FZ20], which is equivalent to a variant of linear datamodels [IPE+22].
- **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 WATERBIRDS $50k$ samples (or models) per learning algorithm; we make a validation split using 10% of these samples.
- 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 [IPE+22], we then re-compute the datamodel on the entire dataset with the optimal λ value and all m training examples.

C.4 Feature transformations

As discussed in Section 2, we counterfactually verify distinguishing features (inferred via human-in-the-loop analysis) by evaluating whether feature transformations change model behavior as hypothesized. Here, we describe the feature transformations used in Section 3 in more detail[7].

Designing feature transformations. We design feature transformations that modify examples by adding a specific patch. We vary the intensity of patch-based transformations via patch size k.Additional details specific to each feature transformation in our case study:

- **Yellow feature.** We add a $k \times k$ square yellow patch to the input.
- **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 [YYF+22] by (a) cropping out human faces from ImageNet validation examples and (b) manually removing mislabeled, low-resolution, and unclear human face images.

Evaluating feature transformations. As shown in Section 3, 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.

C.5 Training infrastructure

Data loading. We use `FFCV`[8] [LIE+22], 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[9]—a SAGA-based GPU solver for ℓ_1-regularized regression—to parallelise datamodel estimation.

[7] The code for these feature transformations is available at `anonymized-url`.

[8] Webpage: `http://ffcv.io`

[9] Github repository: `https://github.com/MadryLab/fast_l1`

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.

D Additional human-in-the-loop analysis

As outlined in Section 2, the second stage of our framework applies human-in-the-loop analysis to infer distinguishing feature transformations from training directions extracted via PCA on residual datamodels. In this section, we present additional human-in-the-loop analysis in order to substantiate the distinguishing features inferred in our case study.

D.1 Tools for Inferring distinguishing features from PCA subpopulations

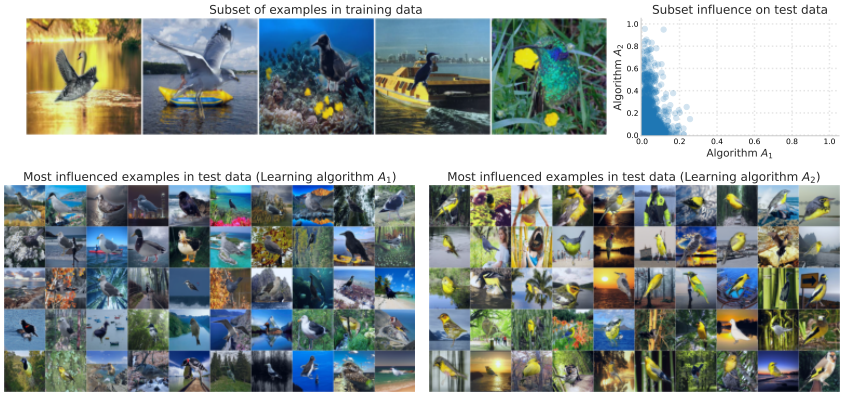
In this section, we outline additional human-in-the-loop tools that we use to analyze subpopulations surfaced by principal components (PCs) of residual datamodels.

- **Class-specific visual inspection.** As shown in Section 3, the subpopulation of test examples whose datamodels have maximum projection onto PCs of residual datamodels largely belong to 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 $\mathcal{A}_2$.

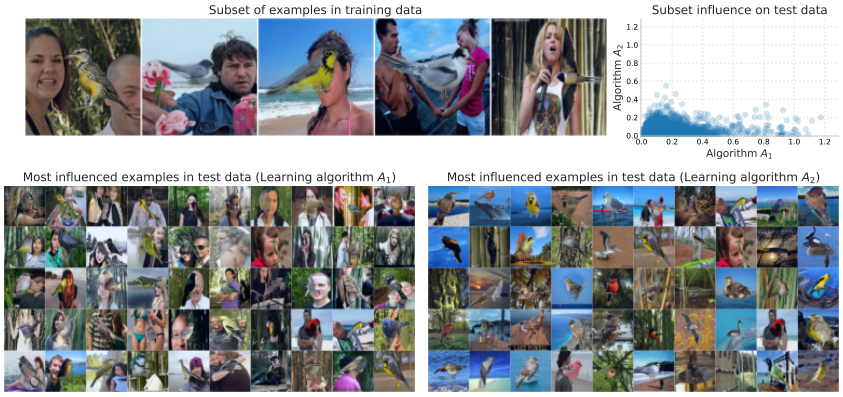
  In our analysis, we (a) identify training examples that have top-most relative influence estimates and then (b) visually contrast the subsets of test examples (one for each learning algorithm) that are most influenced by these training examples.

D.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 4 and additional visual inspection in Figure 5.



(a) "Yellow color" feature



(b) "Human face" feature

Figure 4: **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 2). 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 2). 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 $\mathcal{A}_1$).

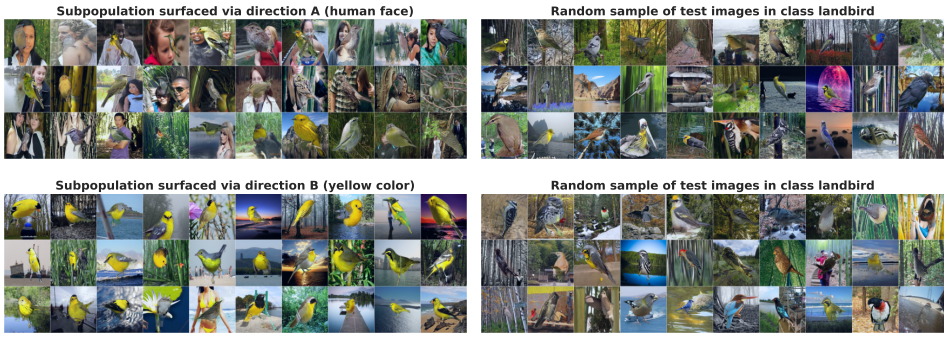
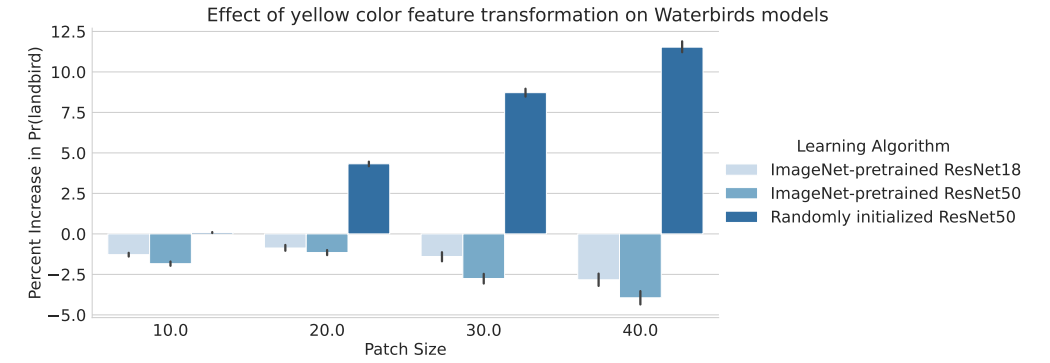


Figure 5: Class-specific visual inspection of WATERBIRDS subpopulations. **(Top)** In contrast to random "landbird" images, the "human face" subpopulation surfaces landbirds with human face(s) in the background. **(Bottom)** Unlike random "landbird" images, the "yellow color" subpopulation surfaces images with yellow birds *or* yellow objects in the background.

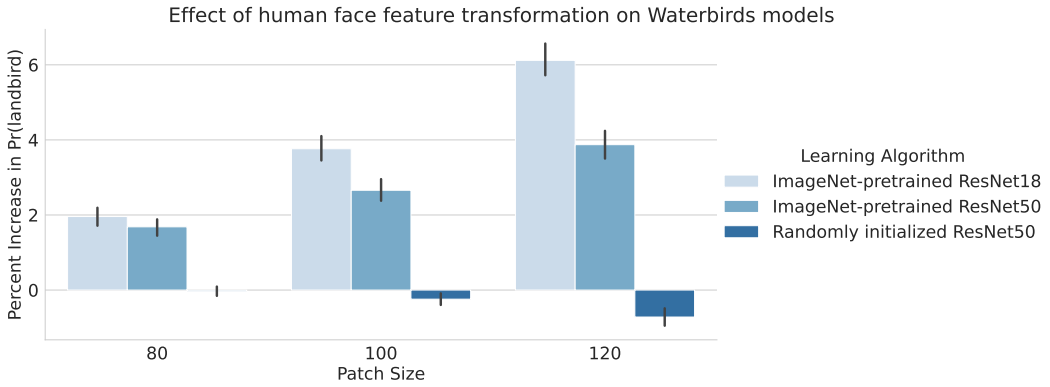
E Additional evaluation of distinguishing feature transformations

Distinguishing feature transformations, which we recall from Section 2, 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 study, we design distinguishing feature transformations that counterfactually verify features that are identified via human-in-the-loop analysis. Our findings in Section 3 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:

In Section 3, 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 6, 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 6: **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 Miscellaneous results

F.1 Aggregate metric for algorithm comparison

As discussed in Appendix A, 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 our WATERBIRDS case study. Figure 7 plots the distribution of cosine similarity of datamodels for multiple learning algorithms over all test examples. The plot shows that ImageNet-pretrained ResNet50 models are, on average, more similar to ImageNet-pretrained ResNet18 models than to ResNet50 models pretrained on synthetically generated data [BWW+21] and models trained from scratch.

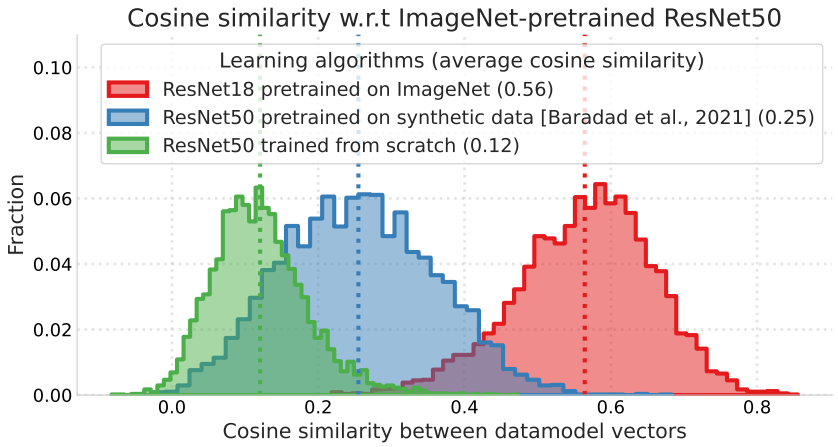


Figure 7: **Datamodel cosine similarity.** We use cosine similarity between two datamodel vectors as an aggregate metric to quantify the similarity of models trained with different learning algorithms. For WATERBIRDS classifiers, datamodels of ImageNet-pretrained ResNet50 and ResNet18 models are more similar to each other than to models pretrained on synthetically generated data and models trained from scratch.

F.2 Explained variance of residual datamodel principal components

Recall from Appendix B 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 $\mathcal{A}_i$. In our case study in Section 3, 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 8 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 $\theta^{(2)}$.

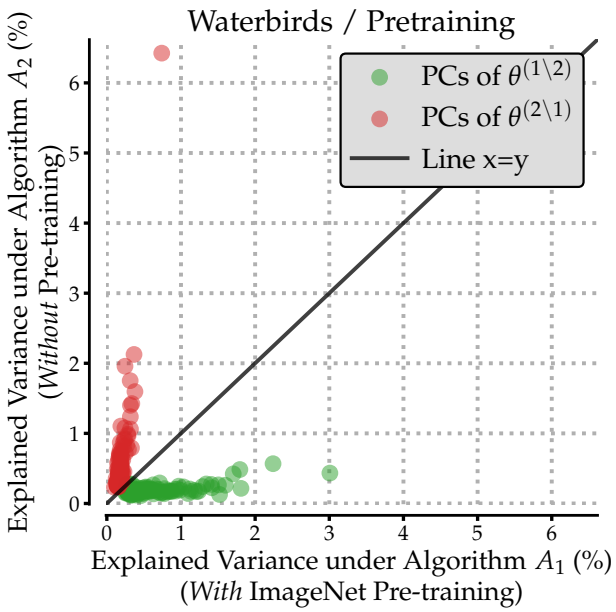


Figure 8: **Explained variance of residual datamodels' principal components.** Highlighted in green (resp. red), the top-100 PCs of residual datamodel $\theta^{(1\setminus 2)}$ (resp. $\theta^{(2\setminus 1)}$) explain a larger (resp. smaller) fraction of datamodel variance under algorithm $\mathcal{A}_1$ than under algorithm $\mathcal{A}_2$.

F.3 Subpopulations surfaced by principal components of residual datamodels

As outlined in Section 2, the human-in-the-loop stage of our framework involves extracting test data subpopulations from 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 3 surface test examples with qualitatively similar properties.

Figure 9: 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 3 analyzes PC #3 (direction **A**).

Figure 10: 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 3 analyzes PC #3 (direction **B**).