Uncovering Hidden Training Dynamics in Neural Networks via Inter-Sample Influence Graphs

Dylan Tan Hong Tai[*1] **Jiayin Zhang**[*2,1] **Rohan Ghosh**[*1] **Mehul Motani**[1]
eledtht@nus.edu.sg suemonsuemon@gmail.com rghosh92@gmail.com motani@nus.edu.sg
National University of Singapore[1], Beijing Institute of Technology[2]

Abstract

Deep learning models are primarily trained through batchwise optimization, where each update can potentially be a tug-of-war among samples, shaping the overall trajectory of learning. Existing interpretability tools, most notably influence functions, have provided valuable insights into how individual training samples affect model predictions, primarily at test time. However, these methods were not intended to capture these inter-sample interactions that arise during training. Here, we ask a complementary question: How does optimizing the loss on one training sample affect the loss on the rest during learning? We introduce Influence Graphs (IGs), directed inter-sample graphs where each edge weight w_{ij} quantifies how optimizing on sample X_i influences the loss of sample X_j. We estimate these influences via simulated batch interventions and slope coefficients of loss changes, enabling scalable construction of IGs during training. We further define the Mean-of-Mean In-Degree Influence (MMDI) and prove it bounds generalization under practical assumptions. Empirically, MMDI correlates strongly with test accuracy in noisy-label settings, making it a useful diagnostic of model quality even before test metrics are available. Finally, we show that IGs reveal distinct, evolving training phases, offering a new lens on the dynamics of learning.

Proceedings of the 29[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

1 INTRODUCTION

Modern deep learning models are trained on massive datasets using stochastic optimization [Krizhevsky et al., 2012], yet our understanding of how individual training samples interact during learning remains lacking. Training data is typically treated as independent and identically distributed, assuming that each sample influences the model in isolation. But this assumption breaks down in practice. Optimization is inherently batchwise, as each update is a negotiation among samples, a tug-of-war where some datapoints pull the model in helpful directions while others introduce conflict or redundancy. Despite this, we lack tools to systematically trace how these interactions unfold.

We argue that this inter-sample interactivity matters. In noisy or imbalanced datasets, a single harmful sample can degrade performance across many others [Arpit et al., 2017]. Conversely, a well-placed example can uplift entire regions of the data manifold. Some samples can act as hubs of influence, shaping the model's behavior far beyond their immediate neighborhood. Others are passive, absorbing structure without contributing much in return. As we show in this work, these dynamics are not just mere observations; they can reveal hidden signals of the generalization performance and robustness of the model. Yet, our current interpretability tools, such as influence functions [Koh and Liang, 2017], focus almost exclusively on test-time behavior. Influence functions primarily inform us which training samples influence predictions [Pruthi et al., 2020], but not how they affect each other during training. Other variants focus on absolute, isolated influences [Feldman, 2020, Yang et al., 2023] of each datapoint based on how much the model changes if it is removed from the training data (leave-one-out). We provide a deeper discussion of related work in Appendix A.

We propose Influence Graphs (IGs) as a framework

[*] Equal contribution.

to fill this void. Influence Graphs are directed inter-sample graphs where each edge weight w_{ij} quantifies the degree to which optimizing the model on sample X_i affects the loss of sample X_j. This formulation allows us to study influence independently of the training trajectory, focusing instead on the causal effect of local updates at a fixed point in model space. By simulating batch interventions and measuring slope-coefficient of loss changes, we isolate these influence relationships and construct a graph that encodes the internal structure of learning and optimization.

Beyond visualizing positively and negatively influential datapoints and edges, this graph can also serve as a diagnostic tool. First, we verify that the IGs are working as intended by testing their ability to prune datasets while preserving generalization performance as a sanity check. We find that IG-based pruning outperforms random pruning and also can potentially yield improvements in performance. Subsequently, we show that aggregate properties of an IG, such as the Mean-of-Mean In-Degree Influence (MMDI), correlate strongly with generalization performance, especially in noisy label settings. This suggests that IGs can serve as early indicators of model quality, even before test metrics are available. Moreover, the evolution of IGs over time reveals distinct phases of training behavior which can be dataset-specific, offering a new way to characterize learning dynamics.

In short, Influence Graphs offer a new primitive for data-centric AI. Rather than limiting interpretability to the behavior of a trained model, they provide a lens on the training process itself, revealing how examples shape one another as learning unfolds. By shifting the focus from model-to-data influence to data-to-data influence, IGs reveal the internal dynamics of optimization beyond static loss or accuracy measures, outlining an approach toward interpretable training dynamics.

Contributions. We outline our contributions.

1. We propose **Influence Graphs (IGs)**, directed graphs over training samples where edge weights quantify how optimizing one sample affects the loss of another.
2. We develop a **slope-coefficient estimator** for IGs, prove it asymptotically recovers the true influence structure under a linear causal model of loss changes, and design a scalable implementation that minimizes redundant computation.
3. We introduce the **Mean-of-Mean In-Degree Influence (MMDI)** and prove it provides a bound on generalization performance under practical assumptions.
4. We show that **MMDI tracks generalization** in noisy-label settings, correlating strongly with test accuracy as label noise increases.
5. We demonstrate that **IGs reveal distinct training phases across datasets**, offering a new lens to study the dynamics of learning.

1.1 Influence Graphs: Methodology

The central question we aim to answer is: *Given a fixed model configuration, what is the impact of further optimizing the model on datapoint i on the loss of datapoint j?* This formulation allows us to study influence independently of the training trajectory, and instead focuses on the causal effect of local updates at a specific model configuration point. In this work we only focus on intra-class influence, where datapoint i and j belong to the same class.

To estimate this influence, we simulate a series of controlled interventions. At each step, we fine-tune the model on a small batch containing datapoint i, measure how this affects the loss of datapoint j, and then reset the model to its original configuration. By repeating this process across many batches and logging the resulting loss changes, we construct a directed graph that captures the influence relationships between datapoints.

This procedure is divided into three stages: simulating batch interventions, measuring loss responses, and constructing the Influence Graph. We also summarize the following in Algorithm 1.

1.2 Simulated Batch Interventions

Given a dataset $S = \{(X_k, Y_k)\}_{k=1}^m$ and a fixed model configuration θ, we estimate how optimizing on one sample influences the loss of another. At each of T mini-iterations, a mini-batch $\mathcal{B}_t$ of size B is drawn, the weights are reset to $\theta_t^{(0)} = \theta$, and K gradient descent steps with learning rate η are applied, yielding an adapted model $\theta_t = \theta_t^{(K)}$. After each adaptation, the model is reset to θ, ensuring all influence estimates are measured relative to the same starting point.

1.3 Measuring Loss Responses

For each datapoint (X_j, Y_j), we compute the change in loss under the adapted model:

$$\delta L_j^{(t)} = L(\theta_t; X_j, Y_j) - L(\theta; X_j, Y_j).$$

1.4 Constructing the Influence Graph

After computing the loss differences, we record pairs $(\overline{\delta L}_i^{(t)}, \overline{\delta L}_j^{(t)})$ for every datapoint $i \in \mathcal{B}_t$ and $j \notin \mathcal{B}_t$, provided that $Y_i = Y_j$. This restriction to same-class

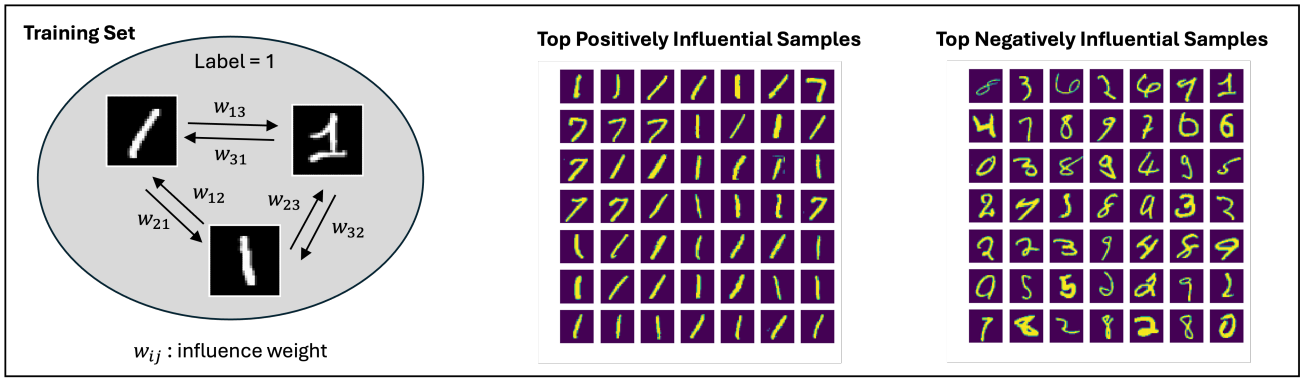


Figure 1: Visualization of Influence Graph and Influential Samples. **Left:** The Influence Graph is constructed by quantifying the influence between data points from the same class. For each pair of data points (i, j), the influence weight w_{ij} between data points i and j represents the positive or negative degree to which the presence of the i^{th} training data point influences the optimization of the j^{th} training data point. **Middle:**. Visualization of samples with highest positive influence in the MNIST dataset. **Right:** Visualization of samples with highest negative influence in the MNIST dataset.

pairs serves two purposes. First, it avoids spurious correlations between datapoints of different classes. Second, it ensures that the resulting Influence Graph remains sparse, which significantly reduces memory and computational overhead.

For each pair (i, j), we incrementally track the sample means, variances, and covariance of the loss differences across all mini-iterations. The covariance between datapoints i and j is defined as: $\text{cov}_{ij} = \frac{1}{n-1}\sum_{t=1}^{n}\left(\delta L_i^{(t)} - \mu_i\right)\left(\delta L_j^{(t)} - \mu_j\right)$, where μ_i and μ_j are the sample means of the loss differences for datapoints $i \in \mathcal{B}_t$ and $j \notin \mathcal{B}_t$, respectively, and n is the number of recorded pairs.

Once all mini-iterations are complete, we compute the slope coefficient between the loss changes of i (inside the batch) and j (outside the batch). This coefficient quantifies the influence of datapoint i on datapoint j, and is given by:

$$w_{ij} = \frac{\text{cov}(\delta L_i, \delta L_j)}{\text{var}(\delta L_i)} \tag{1}$$

where the covariance and variance are computed across all $\delta L_i^{(k)}$ and $\delta L_j^{(k)}$ which denote the loss changes observed in the k-th batch intervention-observation pair. This slope directly captures the causal effect of i on j under the linear structural causal model assumptions.

Remark 1. *Unlike Pearson correlation, this slope coefficient is asymmetric and naturally encodes the directionality of influence from intervened datapoint i to observed datapoint j, consistent with causal interpretation [Pearl, 2009, Spirtes et al., 2000].*

Algorithm 1 Influence Graph Estimation

Require: Dataset $S = \{(X_k, Y_k)\}_{k=1}^{m}$, base weights θ, mini-iterations T, batch size B, inner steps K, learning rate η

Initialize pair-stats for all (i, j) with $Y_i = Y_j$
for $t = 1$ **to** T **do**
  Sample $\mathcal{B}_t \subset S$, $|\mathcal{B}_t| = B$
  $\theta^{(0)} \leftarrow \theta$
  for $k = 1$ **to** K **do**
    $\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta \nabla_\theta L(\theta^{(k-1)}; \mathcal{B}_t)$
  end for
  $\theta_t \leftarrow \theta^{(K)}$
  for all $j = 1, \ldots, m$ **do**
    $\delta L_j^{(t)} \leftarrow L(\theta_t; X_j) - L(\theta; X_j)$
  end for
  for all $i \in \mathcal{B}_t$, $j \notin \mathcal{B}_t$, $Y_i = Y_j$ **do**
    Update means, variances, covariance for (i, j)
  end for
  Reset $\theta^{(k)} \leftarrow \theta$ for next t
end for
Compute final w_{ij} via (1) and form graph $\mathcal{G}$
return $\mathcal{G}$

1.5 Implementation Details

We represent the Influence Graph using classwise blocks, storing statistics only for same-class pairs (i, j) and incrementally tracking counts and second-order moments rather than all raw loss changes. To ensure scalability, we leverage NumPy's in-place operations (e.g., `np.add.at`) for fast, vectorized updates across batches. Each class is represented as a padded block within a larger 3D array, allowing us to accumulate second-order statistics (loss multipliers, passive/active

differences, and their squares) without explicit loops. An inverse lookup table maps local block indices back to global sample IDs, enabling efficient reconstruction of the full graph. The slope-coefficient computations are then computed blockwise and assembled into a sparse COO/CSR matrix, which supports fast storage, retrieval, and downstream analysis. This design allows efficient accumulation of millions of intervention–observation pairs with minimal overhead.

1.6 Theoretical Justification via Linear Structural Causal Models

To justify how the slope-coefficient approach can yield the directed edge weights, we model the relationship between loss changes using a linear structural causal model (SCM). Specifically, our main assumption is that the loss differences of datapoint j, is linearly influenced by the normalized loss change of datapoint i, along with an independent noise term. Proofs are available in Appendix D.

Proposition 1. *Let $\mathcal{B} \subset \{1, \ldots, n\}$ be a randomly sampled batch of datapoints whose loss changes are directly intervened upon. Let δL_i denote the (unnormalized) loss change for $i \in \mathcal{B}$, and δL_j for $j \notin \mathcal{B}$. Assume that for any batch configuration, the loss change of j follows a linear structural causal model (SCM):*

$$\delta L_j = \sum_{i \in \mathcal{B}} \alpha_{ij} \cdot \delta L_i + \varepsilon_j, \tag{2}$$

where $\alpha_{ij} \in \mathbb{R}$ is the causal effect of i on j, and ε_j is zero-mean noise independent of all δL_i. Then, in the limit of infinitely many batch intervention-observation pairs where $i \in \mathcal{B}$ and $j \notin \mathcal{B}$, we have:

$$\alpha_{ij} = \frac{\mathrm{Cov}(\delta L_i, \delta L_j)}{\mathrm{Var}(\delta L_i)} \tag{3}$$

The above result shows that under a linear SCM assumption, the influence weights w_{ij} in (1) correspond to the true path coefficient α_{ij} in a causal graph. Thus, the Influence Graph can be interpreted as a directed causal graph where edge weights reflect estimated causal influence between datapoints during optimization.

Next, we show how measures computed on influence graphs can be related to generalization performance, or more specifically, expected test loss in a leave-one-out setting.

Theorem 1. *Let $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of size $n \geq N$, and let C denote the expected loss of an untrained network over all datapoints. Assume that for batches $\mathcal{B}_j \subset \mathcal{D}$ of size m the linear structural causal model holds:*

$$\delta L_j = \sum_{i \in \mathcal{B}_j} \alpha_{ij}\, \delta L_i + \varepsilon_j, \tag{4}$$

where $|\varepsilon_j| \leq \varepsilon_{\max}$ and $\delta L_i \geq \tau > 0$ for all i. Define the Mean-of-Mean In-Degree Influence (MMDI) with respect to a fixed set $S \subseteq \{1, \ldots, n\}$ as

$$\mathrm{MMDI} := \frac{1}{n|S|} \sum_{j=1}^{n} \sum_{i \in S} \alpha_{ij}, \tag{5}$$

Then, the leave-one-out error can be bounded as:

$$\mathbb{E}_j\big[L((X_j, Y_j); \mathcal{D} \backslash \{(X_j, Y_j)\})\big] \;\leq\; C - \tau \cdot \mathrm{MMDI} + \varepsilon_{\max}.$$

Remark 2. *Theorem 1 bounds leave-one-out error $\mathbb{E}_j\big[L((X_j, Y_j); \mathcal{D} \setminus \{(X_j, Y_j)\})\big]$, when averaged across the training dataset. Leave-one-out error is a well-established and asymptotically consistent estimator of generalization error [Bachmann et al., 2022, Bousquet and Elisseeff, 2002, Rad et al., 2020]. Thus, under the linear SCM assumption on loss changes, MMDI can track generalization error. This assumes that the term C remains unchanged, which represents the expected loss of an untrained model on the datapoints. This assumption may not hold for pre-trained networks, as their expected loss at the start may differ significantly than un-trained network configurations.*

2 EXPERIMENTS

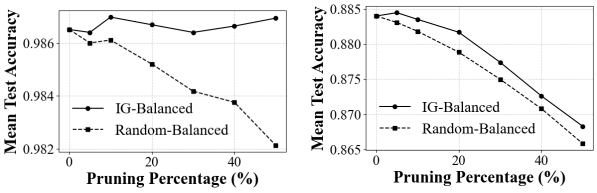
We assess the effectiveness of Influence Graphs (IGs) across four benchmark datasets: MNIST [Deng, 2012], CIFAR-10 [Krizhevsky, 2009], Flowers102 [Nilsback and Zisserman, 2008], and FGVC-Aircraft [Maji et al., 2013]. These datasets cover a spectrum of visual complexity and domains, enabling a robust evaluation of our influence estimation approach.

Our experiments focus on three main objectives. First, we visualize IG structures to reveal positive and negative influence patterns within training data, offering insight into how data samples interact under the model's learned representation. Second, we investigate the correlation between MMDI and test accuracy by analyzing performance across varying label noise levels. Finally, we study the temporal evolution of IGs by generating influence graphs from model checkpoints across training, revealing how influence relationships emerge and stabilize over time. Code is available at github.com/kentridgeai/Influence-Graphs.

Models: To account for dataset-specific characteristics, we employ architectures appropriate to image resolution and complexity: **MNIST / CIFAR-10**:

(a) Top positively influential pairs (b) Top negatively influential pairs

Figure 2: Flowers-102: (a) Top positively influential pairs, (b) Top negatively influential pairs.



(a) MNIST (b) Fashion-MNIST

Figure 3: **Evaluation of IG-guided pruning as a sanity check.** We evaluate the structural integrity of IGs by testing its ability to identify redundant samples. On Fashion-MNIST-5k and MNIST-5k (10-trial average), IG-based pruning consistently outperforms random selection. Notably, on MNIST-5k, the IG effectively filters redundant examples by maintaining full performance even at 50% pruning.

We use a lightweight CNN with three convolutional layers followed by a global average pooling layer, designed for low-resolution datasets. **Flowers102**: We adopt a pretrained VGG-16 model fine-tuned on the dataset, leveraging ImageNet initialization to account for limited sample size and high intra-class variance. **FGVC-Aircraft**: A ResNet-18 architecture is used, comprising four residual stages with two basic blocks each. Its moderate depth balances expressiveness and overfitting control for this fine-grained dataset.

2.1 Visualization of Influences on Flowers-102

We visualize the estimated influences within the Influence Graphs (IGs) by highlighting the most positively and negatively influential data point pairs. These visualizations provide insight into how the model, which is pre-trained on ImageNet, perceives similarity and dissimilarity among examples in the Flowers-102 dataset. For additional visualizations on the rest of the datasets, please refer to Appendix E.

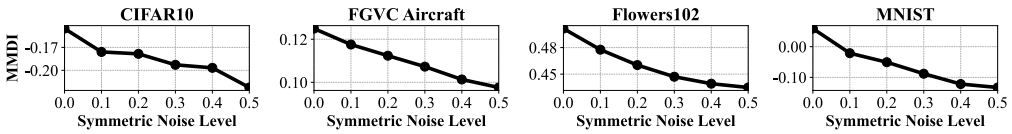
Visualizing Influential Pairs: Figures 2a and 2b show the top positively and negatively influential pairs, respectively, as estimated by the slope-coefficient approach.

Insights from the Visualizations: We observe that many positively influential pairs share high-level semantic similarities despite noticeable low-level differences. These include variations in pose, shear transformations, and brightness, indicating that the pre-trained network is capable of recognizing underlying class-level features even when superficial changes are present. This behavior is likely due to the network's prior exposure to diverse visual patterns during ImageNet pretraining.

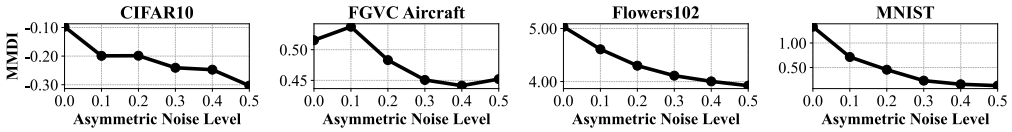
In contrast, negatively influential pairs tend to differ more substantially in both structure and appearance, often involving examples with conflicting visual cues. Together, these visualizations illustrate how IGs expose the semantic structure underlying model behavior, offering an interpretable lens into how the network learns.

2.2 Verifying IG Structure: Dataset Pruning

As a structural sanity check, we evaluate whether the Influence Graph (IG) captures meaningful inter-sample relationships by using it for dataset pruning. The rationale is that a node j with a high mean in-degree influence $\Phi_{in}(j)$ is positively influenced by other samples and is likely redundant:



(a) Symmetric Noise



(b) Asymmetric Noise

Figure 4: **Mean of Mean In-Degree Influence (MMDI)** across four datasets (MNIST, CIFAR-10, Flowers-102, FGVC-Aircraft) under increasing label noise levels from 0% to 50%. (a) shows results under **symmetric noise**, while (b) shows results under **asymmetric noise**. MMDI consistently decreases with higher noise except a few instances, indicating its potential as a diagnostic signal for generalization robustness. Models used: Shallow CNN for MNIST and CIFAR-10, VGG16 for Flowers-102, and ResNet-18 for FGVC-Aircraft.

$$\Phi_{in}(j) = \frac{1}{|\mathcal{D}| - 1} \sum_{i \in \mathcal{D}, i \neq j} w_{ij} \tag{6}$$

If the IG accurately reflects these dependencies, greedily removing nodes with the highest $\Phi_{in}(j)$ should preserve model performance better than random pruning.

Experimental Setup. We test this approach on MNIST-5k and Fashion-MNIST-5k using balanced pruning (removing an equal p% from each class) and report average test accuracy across 10 trials. We emphasize that this is a naive pruning strategy intended solely for structural verification; the objective is not to benchmark against state-of-the-art pruning algorithms. More sophisticated graph-theoretic selection methods, such as those accounting for cluster diversity or centralities, remain an area for future exploration.

Key Findings: Overall, IG-based pruning consistently does better than random pruning. Specifically, in MNIST-5k, test performance is not only maintained, but surpasses the unpruned performance, and stays consistent up to 50% of the data pruned. In Fashion-MNIST however, test performance is only maintained up to 10% pruning, after which we see a decrease in performance. Notably, in both cases, random pruning results in a monotonic decrease in performance from the outset, whereas IG-based pruning preserves or improves accuracy up to a specific pruning threshold. This shows that redundant samples are effectively identified by the IG.

2.3 Can Influence Graphs Indicate Generalizability?

Objective: To explore whether metrics derived from the Influence Graph (IG), particularly the Mean of Mean In-Degree Influence (MMDI), can serve as meaningful indicators of a model's ability to generalize, especially in noisy label settings.

Experimental Setup: We introduced varying levels of symmetric and asymmetric label noise (0% to 50%) across four diverse datasets: MNIST, CIFAR-10, Flowers-102, and FGVC-Aircraft. Corresponding models included a shallow CNN [Lei et al., 2020] for MNIST and CIFAR-10, VGG16 [Simonyan and Zisserman, 2015] for Flowers-102, and ResNet-18 [He et al., 2016] for FGVC-Aircraft.

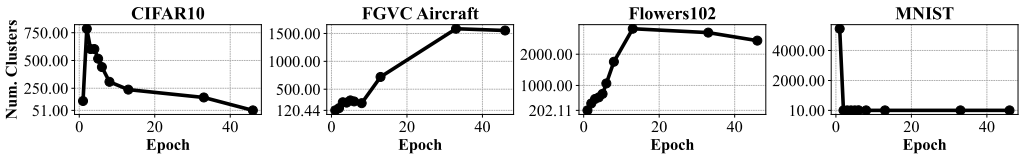
To quantify the influence within each IG, we computed the Mean-of-Mean In-Degree Influence (MMDI):

$$\text{MMDI} := \frac{1}{n|S|} \sum_{j=1}^{n} \sum_{i \in S} w_{ij}, \tag{7}$$

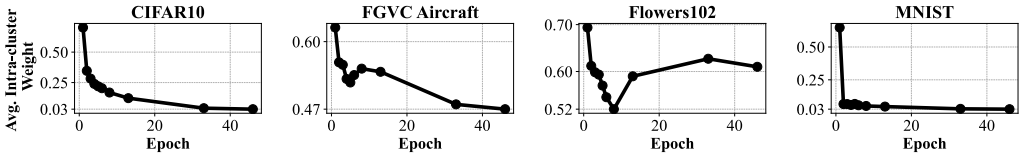
where S represents the set of training samples. Additionally, we assessed the test accuracy of each model on a clean test set. To explore the relationship between MMDI and model performance, we plotted test accuracy versus MMDI (Appendix C).

Key Findings:

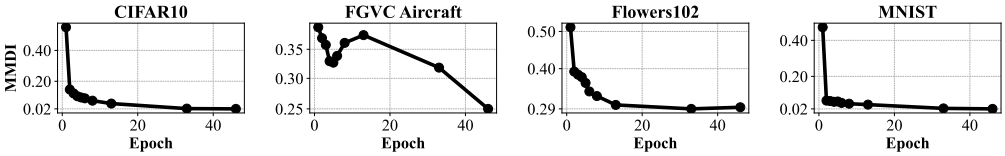
1. MMDI consistently decreases as label noise increases across all datasets, reflecting a disruption in meaningful inter-sample influence.
2. This downward trend occurs under both symmetric and asymmetric noise conditions, with only minor deviations for the asymmetric noise case, which is more



(a) Number of Clusters (averaged over thresholds)

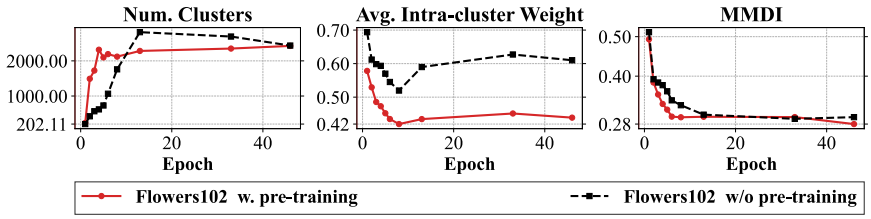


(b) Average Intra-cluster Weight (averaged over thresholds)

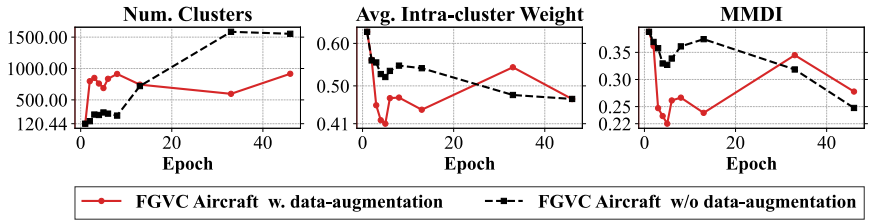


(c) Mean of Mean In-Degree Influence (MMDI)

Figure 5: Epoch-wise evolution of Influence Graph metrics for CIFAR-10, FGVC-Aircraft, Flowers-102, and MNIST datasets. We track the number of clusters (averaged across a fixed set of thresholds), average intra-cluster weight (averaged across a fixed set of thresholds) and Mean of Mean In-Degree Influence (MMDI) across training epochs. Negative weights are removed to focus on positive influence relations. These metrics reveal distinct phases of training dynamics and offer insight into the evolving structure of inter-sample influence.



(a) Flowers102 with and without pre-training



(b) FGVC-Aircraft with and without data augmentation

Figure 6: Epoch-wise evolution of Influence Graph metrics for Flowers102 with and without pre-training, and FGVC-Aircraft with and without data augmentation. We track the number of clusters (computed across a preset range of thresholds), average intra-cluster weight (computed across a preset range of thresholds), and Mean of Mean In-Degree Influence (MMDI) across training epochs.

challenging in general.

3. Lower MMDI correlates with poorer generalization, highlighting its potential as a diagnostic signal for model robustness and dataset quality.

4. When aggregated across datasets, MMDI shows decent correlation with generalization, indicating that it can serve as a signal of generalization performance in general (see Appendix C).

Interpretation: These observations suggest that Influence Graph metrics like MMDI offer a powerful lens to understand and predict generalization performance.

When the graph's structure weakens due to noisy or corrupted labels, the model's capacity to generalize degrades accordingly. Figure 4 visualizes these trends, illustrating how MMDI behavior aligns closely with the impact of label noise on generalization.

2.4 Influence Dynamics Across Training Epochs

Here, we investigate how evolving training dynamics affect influence estimation. A model was trained for m total epochs, and snapshots were captured at intermediate checkpoints $\{n_1, n_2, \ldots, n_k\}$, where each $n_i < m$. These checkpoints represent different stages of the model's learning progression.

For each stored model, we constructed an Influence Graph (IG) following Algorithm 1. This allows us to track how influence relationships evolve over time, providing insight into the stability and dynamics of the internal optimization structure. We evaluate three metrics: (i) number of clusters, (ii) average intra-cluster weight, and (iii) MMDI. The first two metrics are both computed on thresholded IGs over the percentile threshold set $\{10, 20, \ldots, 90\}$. For each thresholded IG, we record the number of connected components and the average intra-cluster edge weight, and then average both quantities over thresholds. Thus, the number of clusters provides a threshold-independent measure of IG fragmentation, while the average intra-cluster weight reflects the strength of influence relations within connected components.

Original Datasets: Figure 5 presents the following metrics plotted across training epochs when networks are trained on the original datasets. Our findings are as follows:

1. **Divergent Convergence Paths:** Optimization dynamics do not follow a universal trajectory. Simpler tasks show a reduction in the number of clusters over time, indicating a transition from localized updates to a more unified, global influence. Complex tasks exhibit the opposite: the influence graph fragments into a higher number of clusters as training progresses, signaling that optimization becomes increasingly confined to specialized sub-groups.

2. **Diminishing Returns in Global Influence:** In datasets that move toward a unified structure (MNIST/CIFAR-10), the average intra-cluster weight decreases alongside the cluster count. This indicates a state of "optimization saturation": as influence spreads across the dataset, its per-sample intensity fades, marking a transition toward a stable equilibrium where further gradient updates yield diminishing returns for the loss landscape.

3. **Persistence of Localized Clusters:** In fine-grained datasets, intra-cluster weights stabilize at non-zero values or exhibit mid-training surges (e.g., Flowers-102 and FGVC-Aircraft). This suggests that even in late-stage training, the optimization process remains concentrated within specific pockets of the dataset. Whether these clusters result from the complexity of the data manifold or from the behaviour of high-capacity network architectures (e.g., ResNet-18) remains an open question.

4. **MMDI as a Progress Marker:** Despite differing cluster dynamics, the Mean-of-Mean In-Degree Influence (MMDI) provides a consistent signature of training progress. In nearly all cases, MMDI decreases monotonically, reflecting the overall exhaustion of learning capacity as the model approaches its final configuration.

5. **Architectural Influence and Optimization Tension:** The discrepancy in Flowers-102, where local interaction strength increases while global MMDI continues to drop, reveals a state of high optimization tension. This implies that while samples within a cluster increasingly support one another, the inter-cluster interactions likely become increasingly antagonistic. This raises the question of whether deeper architectures inherently encourage a "tug-of-war" between global generalization and localized memorization as training progresses.

Data-Augmentation and Pre-Training: We also study the impact of pre-training and data augmentation on Flowers-102 and FGVC-Aircraft. The results are provided in Figure 6. Our objective is to compare the IG trends with and without data-augmentation and pre-training. The overall observations are:

1. **Pre-training as a Structural Catalyst:** On Flowers-102, pre-training slightly accelerates the stabilization of the IG, reaching a steady-state cluster count much earlier than random initialization. Most notably, pre-training yields significantly lower intra-cluster weights throughout optimization despite having a similar MMDI. This suggests that pre-training distributes optimization influence more broadly, preventing the high-intensity "local locking" seen in standard training and maintaining a more cohesive global structure with roughly 500 fewer clusters on average.

2. **Data Augmentation and Manifold Inflation:** On FGVC-Aircraft, data augmentation triggers an immediate spike in localization (higher cluster count) during early epochs, reflecting the initial difficulty of fitting the augmented distribution. Interestingly, it eventually stabilizes at a lower cluster

count than the non-augmented baseline. This suggests that by inflating the class manifolds, augmentation increases sample similarity and effectively delocalizes interactions, preventing the fragmented local memorization typical of static datasets.

3. **Non-linear Capacity Recovery:** Unlike the monotonic MMDI decay in standard settings, augmented training exhibits a sharp initial drop followed by a steady MMDI increase. This indicates a non-linear optimization trajectory where the network, forced to move beyond "lazy" memorization of fixed points, eventually recovers structural influence as it adapts to the broader, inflated manifold of the augmented data.

3 COMPLEXITY ANALYSIS AND EMPIRICAL VERIFICATION

The computational cost of our method is optimized by restricting forward-pass recomputations and graph updates to relevant sub-groups. We consider one iteration to be a set of mini-iterations that cover the whole dataset once. Let N be the total number of samples and C be the number of classes. The total runtime $T(N, C)$ per iteration decomposes as:

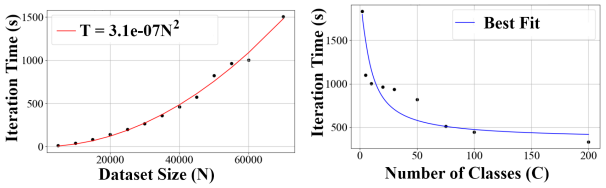
1. **Forward-pass recomputation:** We only recompute the model's forward pass for samples from the classes present in a batch of size b. The number of distinct classes in a batch is roughly $\mathbb{E}[L] \approx C[1-(1-1/C)^b]$. Assuming roughly N/C samples per class, total cost across N/b batches is $\approx \frac{N}{b} \cdot \frac{N}{C} \cdot C[1-(1-1/C)^b] = \frac{N^2}{b} C[1-(1-1/C)^b]$.
2. **Intra-class graph updates:** For each batch, we must update pairwise relationships within the sampled classes. Restricting Influence Graph updates to intra-class pairs reduces the naive $O(N^2)$ update to $\frac{N}{b} \cdot b \cdot \frac{N}{C} = \frac{N^2}{C}$.

Combining them, overall runtime per iteration is:

$$T(N, C) \approx \alpha \cdot \frac{N^2}{b}\left[1-(1-1/C)^b\right] + \beta \cdot \frac{N^2}{C} + \gamma \quad (8)$$

where α is a proportionality constant for the forward pass and γ accounts for constant overhead.

Empirical Validation. We verify this model on MNIST ($N = 60$k) using a single NVIDIA A100 GPU. As shown in Figure 7(a), the combined model captures the overall iteration time complexity trajectory w.r.t the number of classes C. Furthermore, when varying N at fixed C, a power-law fit $T(N) \propto N^v$ yields $v \approx 2$, confirming the quadratic scaling with dataset size (Figure 7(b)).



(a) Iteration Time v/s N (b) Iteration Time v/s C

Figure 7: **Runtime verification**. (a) Scaling with classes C follows the predicted combined model. (b) Scaling with size N confirms $O(N^2)$ growth.

Scalability. Table 1 reports absolute runtimes for 10 iterations (our default). While the $O(N^2)$ scaling suggests challenges for massive datasets, the $1/C$ dependence provides a significant mitigative factor as class counts grow. For instance, on ImageNet ($C = 1000$), the increased granularity of classes further reduces the per-class recomputation and update load, suggesting a viable path for future scaling.

Dataset	Time (s)	Dataset	Time (s)
MNIST	10010.1	Flowers-102	1171.3
CIFAR-10	9765.5	FGVC-Aircraft	2179.8

Table 1: Total runtime for 10 iterations (A100 GPU).

4 REFLECTIONS

Our experiments show that Influence Graphs (IGs) constructed via the slope-coefficient approach effectively capture inter-sample relationships and are sensitive to both data quality and model dynamics. On Flowers-102, IGs reveal semantic similarities and dissimilarities among samples despite large visual variation, highlighting the interpretability of influence patterns. Our implementation keeps IG estimation computationally tractable and avoids redundant computation of losses. The strong correlation between MMDI and test accuracy under varying label noise suggests that IGs can serve as reliable indicators of generalization and robustness. Temporal analysis across training epochs reveals that optimization can unfold in very different ways depending on the dataset: in some cases models gradually build broad, global influence, while in others they fragment into smaller, localized pockets of interaction. Interestingly, pre-training and data augmentation can significantly reshape these dynamics, sometimes producing unexpected non-linear training trajectories which take surprising detours before settling into equilibrium. Overall, IGs emerge as a valuable lens into the hidden mechanics of training, offering fresh insight into model robustness, learning dynamics, and optimization behavior.

ACKNOWLEDGEMENTS

This research is supported by A*STAR, CISCO Systems (USA) Pte. Ltd and the National University of Singapore under its Cisco-NUS Accelerated Digital Economy Corporate Laboratory (Award I21001E0002). We also thank the Kent-Ridge AI research group at the National University of Singapore for helpful discussions.

References

D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, A. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien. A closer look at memorization in deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

G. Bachmann, T. Hofmann, and A. Lucchi. Generalization through the lens of leave-one-out error. In *International Conference on Learning Representations*, 2022.

O. Bousquet and A. Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar): 499–526, 2002.

L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

V. Feldman. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd annual ACM SIGACT symposium on theory of computing*, pages 954–959, 2020.

A. Ghorbani and J. Zou. Data shapley: Equitable valuation of data for machine learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2242–2251, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

R. Jia, D. Dao, B. Wang, F. Hubis, N. Hynes, N. M. Gürel, B. Li, C. Zhang, D. Song, and C. Spanos. Towards efficient data valuation based on the shapley value. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1167–1176, 2019.

P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

F. Lei, X. Liu, Q. Dai, and B. W.-K. Ling. Shallow convolutional neural network for image classification. *SN Applied Sciences*, 2(97), 2020.

S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 947–954, 2013.

M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, pages 722–729, 2008.

J. Pearl. *Causality*. Cambridge university press, 2009.

G. Pruthi, F. Liu, S. Kale, and M. Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

K. R. Rad, W. Zhou, and A. Maleki. Error bounds in estimating the out-of-sample prediction error using leave-one-out cross validation in high-dimensions. In *International Conference on Artificial Intelligence and Statistics*, pages 4067–4077. PMLR, 2020.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT press, 2000.

M. Toneva, A. Sordoni, R. T. des Combes, A. Trischler, Y. Bengio, and G. J. Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019.

S. Yang, Z. Xie, H. Peng, M. Xu, M. Sun, and P. Li. Dataset pruning: Reducing training data by examining generalization influence. In *The Eleventh International Conference on Learning Representations*, 2023.

C.-K. Yeh, J. Kim, I. E.-H. Yen, and P. K. Ravikumar. Representer point selection for explaining deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, 2018.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Supplementary Materials for Uncovering Hidden Training Dynamics in Neural Networks via Inter-Sample Influence Graphs

A RELATED WORK AND BACKGROUND

A.1 Classical Influence Functions

Influence functions originate in robust statistics as tools to quantify the effect of infinitesimal perturbations to the data distribution on an estimator. In modern machine learning, they are used to estimate how up-weighting or removing a single training sample affects a model's parameters or a test loss. A typical formulation considers the empirical risk minimizer $\hat{\theta}$ and studies the effect of up-weighting a point (x_i, y_i) by a small ϵ, leading to a parameter perturbation approximated via the inverse Hessian:

$$\frac{d\hat{\theta}_\epsilon}{d\epsilon}\Big|_{\epsilon=0} \approx -H_{\hat{\theta}}^{-1} \nabla_\theta \ell(x_i, y_i; \hat{\theta}), \tag{9}$$

$$\mathcal{I}_{\text{up, loss}}(i, \text{test}) \approx -\nabla_\theta \ell(x_{\text{test}}, y_{\text{test}}; \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta \ell(x_i, y_i; \hat{\theta}). \tag{10}$$

Leave-one-out influence is approximated by integrating this infinitesimal up-weighting to removal, yielding a first-order estimate of how dropping a point would alter parameters and downstream loss. These approximations are popular because they avoid retraining, reduce computational cost, and provide principled, differentiable sensitivity analyses tied to the local curvature captured by $H_{\hat{\theta}}^{-1}$.

A.2 Representative Applications

[Koh and Liang, 2017] introduced influence functions to deep learning, tracing which training points most affect a given prediction. Their method has been used for debugging mislabeled samples, data sanitization, and interpretability of test-time predictions. [Pruthi et al., 2020] proposed TracIn, which approximates per-example influence by summing gradient inner products across checkpoints, bypassing Hessian inversion. [Yeh et al., 2018] developed Representer Point Selection, decomposing predictions into contributions from training points. Other works such as Data Shapley [Ghorbani and Zou, 2019] and Data Valuation [Jia et al., 2019] adopt cooperative-game-theoretic approaches to assign scalar values to training points.

All of these methods quantify how single samples affect test-time behavior or global utility. They do not model the directed, pairwise influence between training samples during the learning process, nor do they reveal the evolving inter-sample dynamics across training phases. Our Influence Graphs instead measure and encode sample-to-sample causal effects via local interventions, enabling analysis of training-time "tug-of-war" dynamics and enhancing the interpretability of the training process itself.

A.3 Memorization and Dataset Pruning

[Feldman, 2020] showed that generalization in overparameterized models often relies on memorizing rare "long-tail" examples, analyzing per-example contributions to generalization. Their work highlights the importance of individual samples but focuses on scalar importance rather than relational structure.

[Yang et al., 2023] proposed an approach to dataset pruning which estimates each single point's generalization influence to prune datasets while preserving accuracy. Their approach also provides per-point scalar influence estimates to guide pruning.

Both of these works treat samples as isolated units with scalar influence on generalization or utility. In contrast, Influence Graphs focuses on how samples affect each other's losses during optimization, making training-time interactions the priority and enabling various graph-level analyses in addition to tracking generalization performance.

A.4 Other Interpretability Uses

Influence-function frameworks have also been applied to debugging and outlier detection, robustness and fairness auditing, and data quality analysis. For example, [Toneva et al., 2019] introduced "forgetting events," tracking when samples flip from correct to incorrect during training. While complementary, these approaches do not provide causal pairwise influence or graph-based structure.

In summary, prior work interprets model predictions or assigns scalar values to training points. Our approach is not interpretability of a trained model per se; it is an interpretable view of the *training process*, revealing the loss interactions between training samples through a directed graph of causal influences. This shift from model-to-data influence to data-to-data influence reveals structure in optimization dynamics beyond static loss or accuracy, and supports early, training-time diagnostics of generalization.

B TIME COMPLEXITY ANALYSIS AND EMPIRICAL VERIFICATION

B.1 Complexity Discussion

Our method has been carefully optimized and streamlined to reduce redundant computation and exploit structural properties of the problem. The total runtime per iteration can be decomposed into two main parts:

- **Forward-pass recomputation:** Only a subset of examples need to be recomputed, specifically those belonging to the distinct classes sampled in a mini-batch. This reduces the cost from a naive $O(N^2)$ recomputation to a term proportional to the expected number of distinct classes per batch.
- **Graph updates:** Pairwise updates are restricted to intra-class pairs, which scales as $O(N^2/C)$ rather than the full $O(N^2)$.

Through these optimizations, the overall complexity per iteration is:

$$T(N, C) \approx \frac{KN^2}{b} P \cdot \left[1 - (1 - 1/C)^b\right] + \frac{KN^2}{C},$$

where N is the dataset size, C the number of classes, b the batch size, K the number of iterations, and P a proportionality constant reflecting forward-pass cost.

B.2 Derivation Details

We now provide a more detailed derivation of the per-iteration complexity. The key observation is that our algorithm avoids redundant computation by (i) restricting the forward-pass of models to only the datapoints having the same labels as the current batch, and (ii) restricting graph updates to intra-class pairs. This leads to a significant reduction compared to the naive $O(N^2)$ cost.

Expected number of distinct classes. For a batch of size b sampled with replacement from C classes, the expected number of distinct classes is (roughly)

$$\mathbb{E}[L] = C\left(1 - \left(1 - \tfrac{1}{C}\right)^b\right).$$

This quantity grows sublinearly with C and saturates quickly, reflecting the fact that only a limited number of classes are represented in each batch.

Forward-pass recomputation term. Each distinct class in the batch requires recomputation of its forward pass. Since there are $\frac{N}{C}$ examples per class on average, and $\frac{N}{b}$ batches per epoch, the total recomputation cost is

$$\frac{N}{b} \cdot \frac{N}{C} \cdot \mathbb{E}[L] \approx \frac{N^2}{b} \cdot \left[1 - (1 - 1/C)^b\right].$$

This term decreases as C increases, since more classes reduce the per-class load. The factor $[1-(1-1/C)^b]$ captures the probability that a given class appears at least once in a batch.

Graph update term. For each batch, we must update pairwise relationships within the sampled classes. Restricting to intra-class pairs reduces the naive $O(N^2)$ cost to

$$\frac{N}{b} \cdot b \cdot \frac{N}{C} \;=\; \frac{N^2}{C}.$$

Combined complexity. Adding the two contributions yields the final per-iteration complexity:

$$T(N,C) \;\approx\; \frac{N^2}{b} \cdot \Big[1-(1-1/C)^b\Big] \;+\; \frac{N^2}{C}.$$

B.3 Experimental Verification

The following experiments to test the runtimes have been done on the full MNIST dataset (60k samples).

B.3.1 Scaling with Number of Classes C

We varied the number of classes C while keeping N fixed. To test the theoretical decomposition, we compared three candidate models for the runtime:

1. A model with only the factor term and an offset:

$$T(C) \;\approx\; \alpha \cdot \Big(1-(1-1/C)^b\Big) + \gamma,$$

2. A model with only the $1/C$ term and an offset:

$$T(C) \;\approx\; \frac{\beta}{C} + \gamma,$$

3. A combined model with both terms and an offset:

$$T(C) \;\approx\; \alpha \cdot \Big(1-(1-1/C)^b\Big) + \frac{\beta}{C} + \gamma.$$

Results are shown in Figure 8(a). Among these alternatives, the combined model provided the best fit to the empirical runtimes. This confirms that both the "factor" contribution (from distinct classes per batch) and the $1/C$ contribution (from intra-class graph updates) are necessary to accurately capture the observed scaling.

B.3.2 Scaling with Dataset Size N

We next varied N while fixing C. The runtimes were fit using a power-law model of the form

$$T(N) \;\approx\; a \cdot N^v.$$

Results are shown in Figure 8(b). The best-fit exponent was found to be $v \approx 2$, confirming that the runtime grows quadratically with dataset size. This is consistent with the $O(N^2)$ complexity predicted by the graph update term in the theoretical analysis.

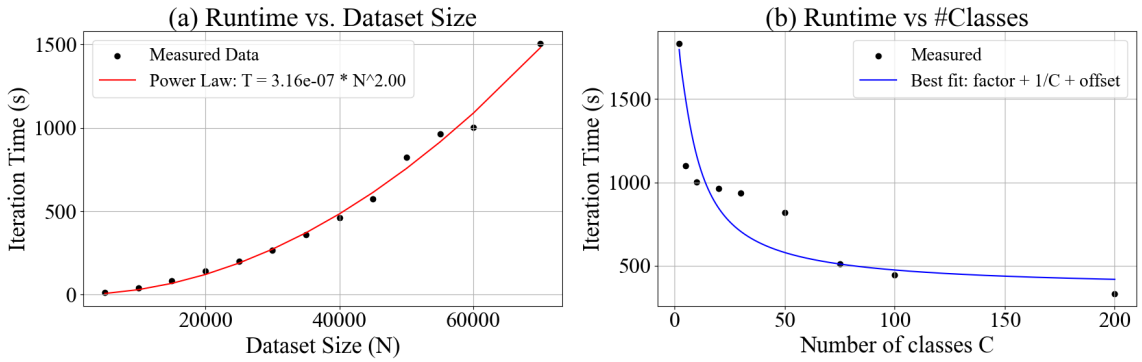


Figure 8: Empirical verification of runtime complexity (single iteration). (a) Runtime vs. number of classes C (MNIST, $N = 60000$, $b = 10$). The best fit is given by the combined model (factor + $1/C$ + offset). (b) Runtime vs. dataset size N (fixed $C = 10$). A power-law fit yields an exponent of ≈ 2, confirming $O(N^2)$ scaling.

B.4 Discussion

The experiments validate the theoretical complexity analysis:

- Varying C confirms the factor + $1/C$ + offset model, showing that our optimizations yield strictly decreasing runtime with more classes.
- Varying N confirms the quadratic $O(N^2)$ scaling, consistent with the graph update term.

At first glance, the $O(N^2)$ dependence may appear prohibitive for very large datasets such as ImageNet, which contains over one million images. However, it is important to note that the number of classes C also increases substantially in such datasets (e.g., $C = 1000$ for ImageNet), and thus it is possible to extend our analysis to larger datasets with more classes such as Imagenet for future extensions of this work.

C CORRELATION BETWEEN MMDI AND TEST ACCURACY

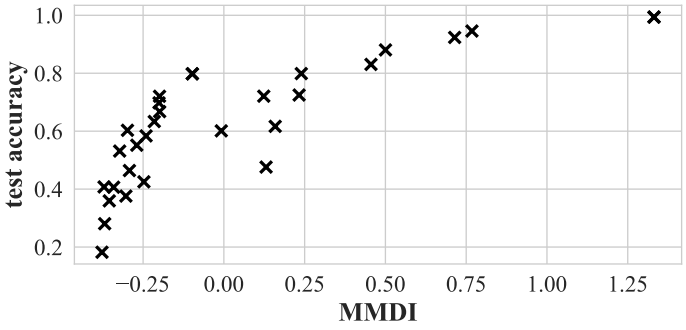


Figure 9: Test accuracy vs. MMDI across MNIST, CIFAR-10, and FGVC-Aircraft under symmetric and asymmetric label noise. Each point is a training run; higher MMDI aligns with higher accuracy, with a Spearman correlation of 0.91. Flowers-102 is excluded due to pretraining bias in C (Theorem 1).

We now study the empirical correlation between the Mean-of-Mean In-Degree Influence (MMDI) and test accuracy across datasets and noise settings. We consider MNIST, CIFAR-10, and FGVC-Aircraft under both symmetric and asymmetric label noise. We exclude Flowers-102 because its noisy-label experiments rely on ImageNet-pretrained networks, which introduce a bias in the constant C (the untrained loss) appearing in Theorem 1. Since C is shifted by pretraining, the resulting MMDI values are not directly comparable.

C.1 From MMDI to Accuracy

We operate under the simplifying assumption that correctly classified samples incur zero loss and misclassified samples incur loss $\log K$ (uniform prediction over K classes). In this scenario, the expected test loss is

$$L \;=\; (1 - A)\,\log K, \tag{11}$$

where A is the test accuracy. Inverting gives

$$A \;=\; 1 - \tfrac{L}{\log K}. \tag{12}$$

By Theorem 1, $L \le C - \tau m \cdot \mathrm{MMDI} + \varepsilon_{\max}$, so

$$A \;\gtrsim\; 1 - \frac{C - \tau m \cdot \mathrm{MMDI} + \varepsilon_{\max}}{\log K}. \tag{13}$$

Moreover, since τ is the minimum per-sample loss, it is natural to take $\tau \propto \log K$ (i.e., proportional to the loss range), which yields cancellation of constants when composing loss bounds with accuracy, making the accuracy–MMDI relation stable across equidistributed settings.

C.2 Empirical Correlation

Across MNIST, CIFAR-10, and FGVC-Aircraft, we compute the Spearman correlation between MMDI and test accuracy under both symmetric and asymmetric label noise. The resulting correlation is 0.91, demonstrating that MMDI is a reliable predictor of generalization performance, even before test labels are available.

D PROOFS OF THEORETICAL RESULTS

Proposition 2. *Let $\mathcal{B} \subset \{1, \ldots, n\}$ be a randomly sampled batch of datapoints whose loss changes are directly intervened upon. Let δL_i denote the (unnormalized) loss change for $i \in \mathcal{B}$, and δL_j for $j \notin \mathcal{B}$. Assume that for any batch configuration, the loss change of j follows a linear structural causal model (SCM):*

$$\delta L_j = \sum_{i \in \mathcal{B}} \alpha_{ij} \cdot \delta L_i + \varepsilon_j, \tag{14}$$

where $\alpha_{ij} \in \mathbb{R}$ is the causal effect of i on j, and ε_j is zero-mean noise independent of all δL_i. Assume across all batch configurations that the update δL_i is independent of the batch-position update δL_k for any fixed position k within the batch $\mathcal{B}$. Then, in the limit of infinitely many batch intervention-observation pairs where $i \in \mathcal{B}$ and $j \notin \mathcal{B}$, we have:

$$\alpha_{ij} = \frac{\mathrm{Cov}(\delta L_i, \delta L_j)}{\mathrm{Var}(\delta L_i)} \tag{15}$$

Remark 3. *We've updated the result here with an additional independence assumption. Because batches are sampled at random and the non-i indices permute across experiments, the identity of the k-th co-member within a batch $\mathcal{B}$ is effectively exchangeable over the remaining datapoints (excluding j which is outside $\mathcal{B}$ each time). Under this symmetry and random batch selection, it is reasonable to assume that the fixed update δL_i is independent of the update δL_k at a given batch position k, across all batch configurations.*

Proof. By the structural causal model we have

$$\delta L_j \;=\; \alpha_{ij}\,\delta L_i \;+\; \sum_{k \in \mathcal{B} \setminus \{i\}} \alpha_{kj}\,\delta L_k \;+\; \varepsilon_j.$$

Taking covariance with δL_i gives

$$\mathrm{Cov}(\delta L_i, \delta L_j) = \mathrm{Cov}(\delta L_i, \alpha_{ij}\delta L_i) + \mathrm{Cov}\Big(\delta L_i, \sum_{k \in \mathcal{B} \setminus \{i\}} \alpha_{kj}\delta L_k\Big) + \mathrm{Cov}(\delta L_i, \varepsilon_j).$$

The first term equals $\alpha_{ij}\,\mathrm{Var}(\delta L_i)$. By the independence assumption across batch configurations, $\mathrm{Cov}(\delta L_i, \delta L_k) = 0$ for any fixed batch position k, so the second term vanishes. Since ε_j is independent of δL_i and has mean zero, the third term also vanishes. Therefore

$$\mathrm{Cov}(\delta L_i, \delta L_j) \;=\; \alpha_{ij}\,\mathrm{Var}(\delta L_i).$$

Rearranging yields

$$\alpha_{ij} \;=\; \frac{\mathrm{Cov}(\delta L_i, \delta L_j)}{\mathrm{Var}(\delta L_i)}.$$

□

Theorem 2. *Let $\mathcal{D} = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of size $n \geq N$, and let C denote the expected loss of an untrained network over all datapoints. Assume that for batches $\mathcal{B}_j \subset \mathcal{D}$ of size m the linear structural causal model holds:*

$$\delta L_j = \sum_{i \in \mathcal{B}_j} \alpha_{ij}\,\delta L_i + \varepsilon_j, \tag{16}$$

where $|\varepsilon_j| \leq \varepsilon_{\max}$ and $\delta L_i \geq \tau > 0$ for all i. Define the Mean-of-Mean In-Degree Influence (MMDI) with respect to a fixed set $S \subseteq \{1, \ldots, n\}$ as

$$\mathrm{MMDI} := \frac{1}{n^2} \sum_{j=1}^{n} \sum_{i \in S} w_{ij}, \tag{17}$$

where $w_{ij} = \alpha_{ij}$ is the (population) slope coefficient from i to j. Then:

$$\mathbb{E}_j\left[L((X_j, Y_j); \mathcal{D} \setminus \{(X_j, Y_j)\})\right] \leq C \;-\; \tau\, m \cdot \mathrm{MMDI} + \varepsilon_{\max}. \tag{18}$$

Proof. For a fixed j, let C_j be the untrained loss on (X_j, Y_j). After training on all but (X_j, Y_j),

$$L((X_j, Y_j); \mathcal{D} \setminus \{(X_j, Y_j)\}) \;=\; C_j - \delta L_j.$$

Taking expectation over the random batch and using $\delta L_i \geq \tau$ and $|\varepsilon_j| \leq \varepsilon_{\max}$,

$$\mathbb{E}_{\mathcal{B}_j}[\delta L_j] \;=\; \mathbb{E}_{\mathcal{B}_j}\left[\sum_{i \in \mathcal{B}_j} \alpha_{ij}\,\delta L_i\right] + \mathbb{E}_{\mathcal{B}_j}[\varepsilon_j] \;\geq\; \tau \cdot \mathbb{E}_{\mathcal{B}_j}\left[\sum_{i \in \mathcal{B}_j} \alpha_{ij}\right] \;-\; \varepsilon_{\max}.$$

By uniform sampling, each $i \in S$ is included in $\mathcal{B}_j$ with probability $p = m/n$, hence

$$\mathbb{E}_{\mathcal{B}_j}\left[\sum_{i \in \mathcal{B}_j \cap S} \alpha_{ij}\right] \;=\; p \sum_{i \in S} \alpha_{ij}.$$

Therefore,

$$\mathbb{E}_{\mathcal{B}_j}[L((X_j, Y_j); \mathcal{D} \setminus \{(X_j, Y_j)\})] \;\leq\; C_j \;-\; \tau\,\frac{m}{n} \sum_{i \in S} \alpha_{ij} \;+\; \varepsilon_{\max}.$$

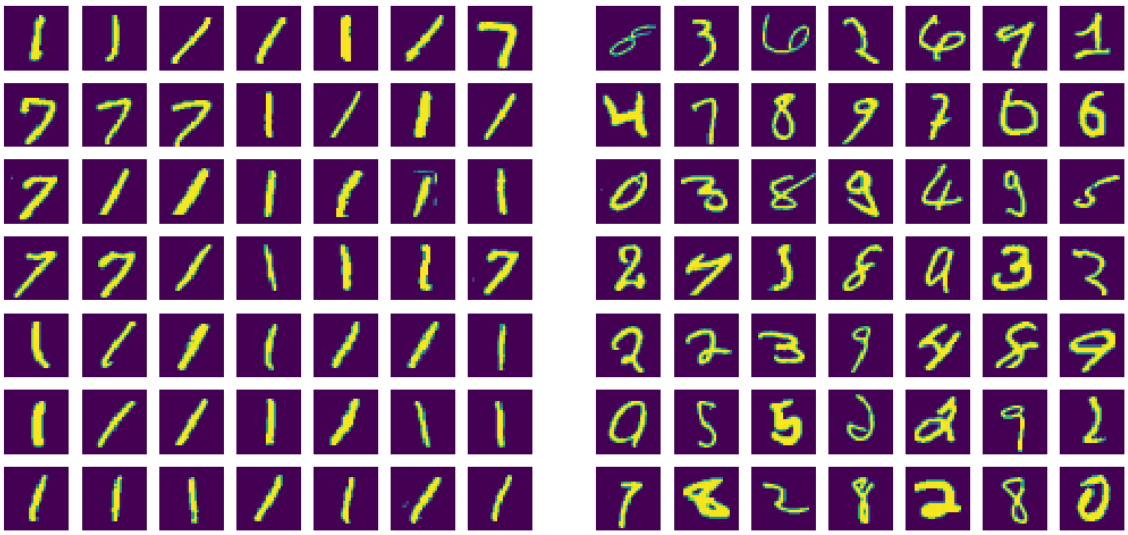
Averaging over $j = 1, \ldots, n$ gives

$$\begin{aligned}\mathbb{E}_j\,\mathbb{E}_{\mathcal{B}_j}[L((X_j, Y_j); \mathcal{D} \setminus \{(X_j, Y_j)\})] &\leq \frac{1}{n}\sum_{j=1}^{n} C_j \;-\; \tau\, m \cdot \frac{1}{n^2} \sum_{j=1}^{n} \sum_{i \in S} \alpha_{ij} \;+\; \varepsilon_{\max} \\ &= C \;-\; \tau\, m \cdot \mathrm{MMDI} \;+\; \varepsilon_{\max},\end{aligned}$$

which completes the proof. □

E MORE INFLUENCE VISUALIZATIONS

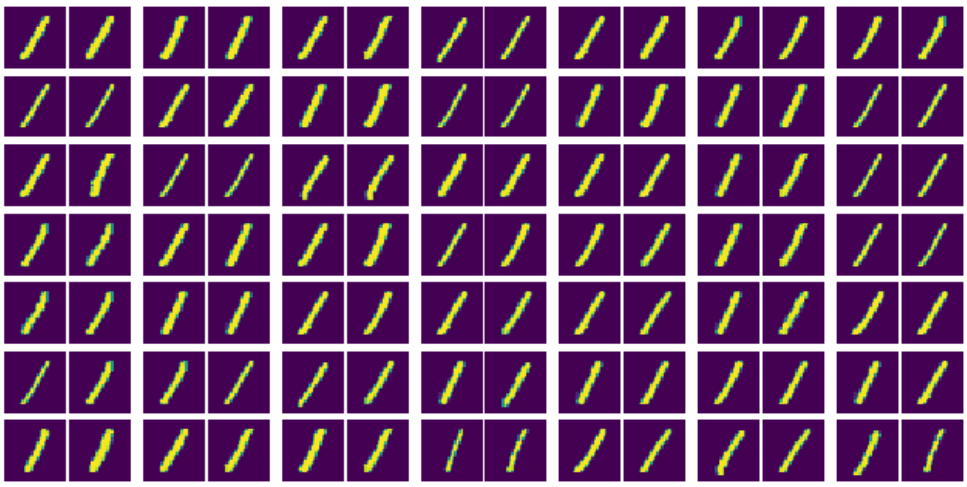
In the subsequent figures, we present additional visualizations of the computed influence measures, specifically highlighting the most positively and negatively influential pairs and nodes for MNIST, CIFAR-10, and FGVC-Aircraft. Please note that all figures are organized based on the influence measures, with the highest/lowest values of influence positioned at the top-left, and influence values increasing progressively along each row and subsequently to the next column.



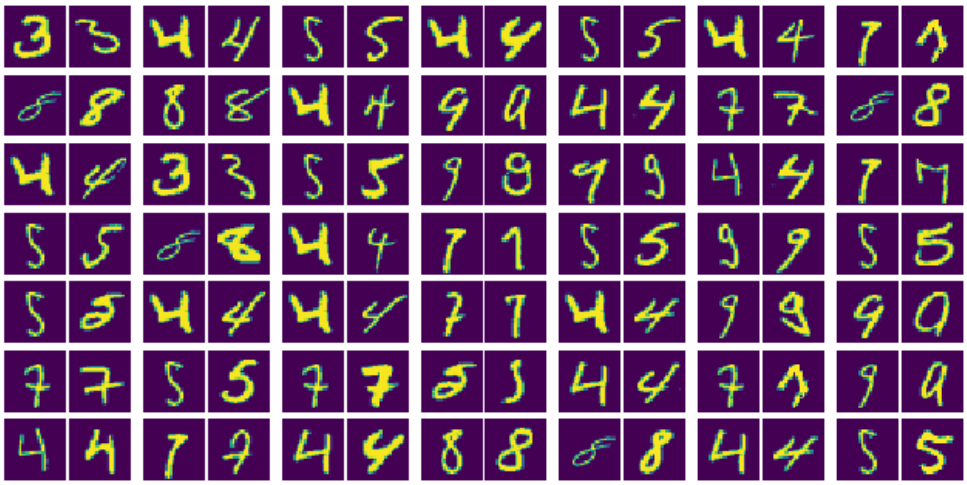
(a) Top positively influential nodes.

(b) Top negatively influential nodes.

Figure 10: MNIST: (a) Nodes with highest positive influence, (b) Nodes with highest negative influence.

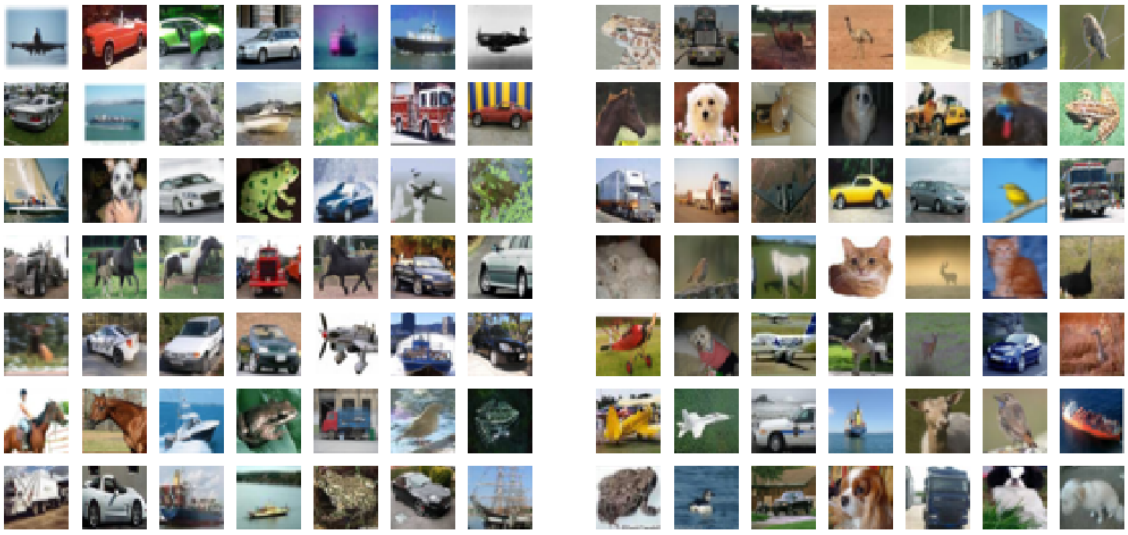


(a) Top positively influential pairs



(b) Top negatively influential pairs

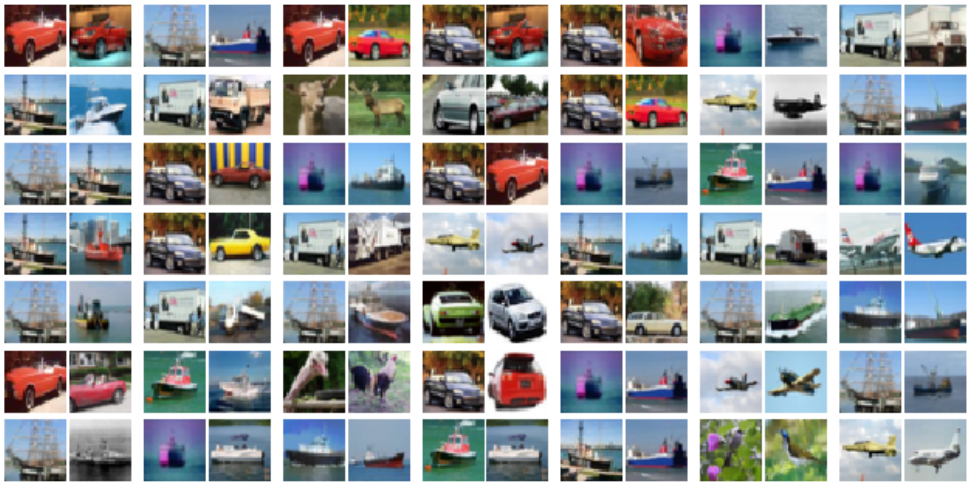
Figure 11: MNIST: (a) Top positively influential pairs, (b) Top negatively influential pairs.



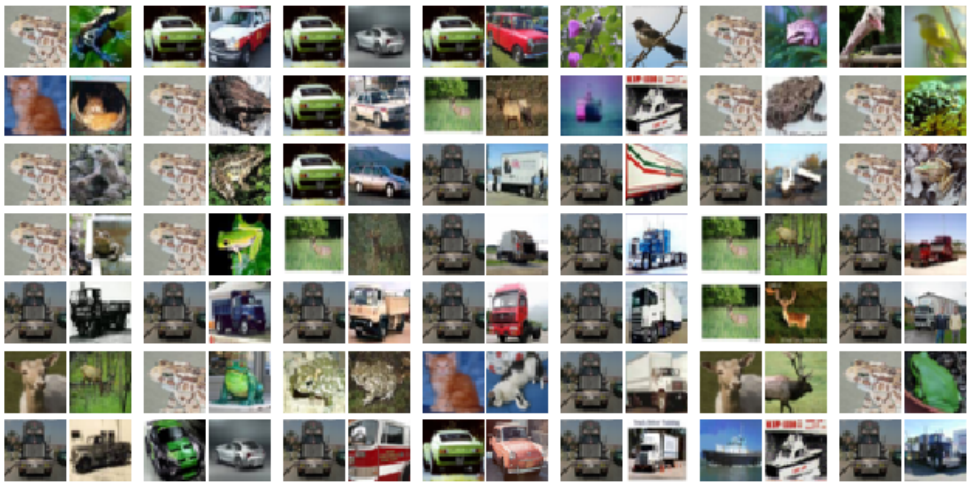
(a) Top positively influential nodes

(b) Top negatively influential nodes

Figure 12: CIFAR10: (a) Top positively influential nodes, (b) Top negatively influential nodes.

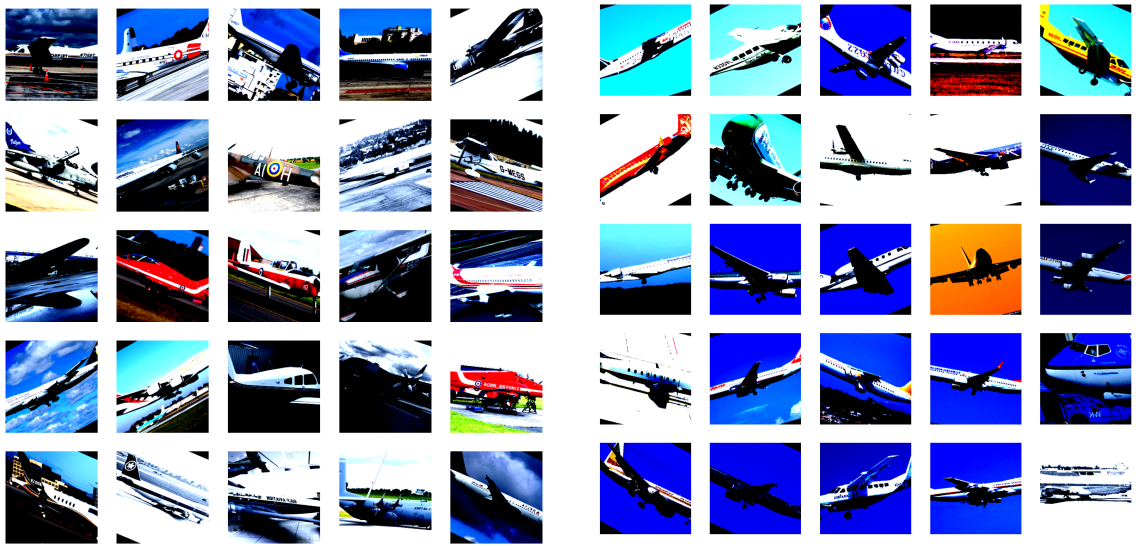


(a) Top positively influential pairs.



(b) Top negatively influential pairs.

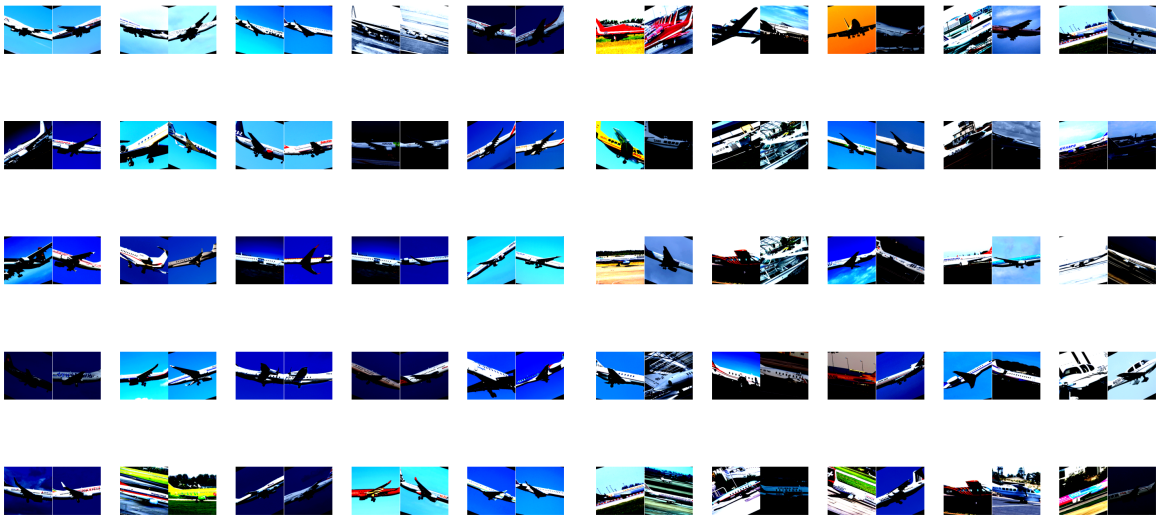
Figure 13: CIFAR10: (a) Pairs with highest positive influence, (b) Pairs with highest negative influence.



(a) Top positively influential nodes

(b) Top negatively influential nodes

Figure 14: FGVC-Aircraft: (a) Top positively influential nodes, (b) Top negatively influential nodes.



(a) Top positively influential pairs.

(b) Top negatively influential pairs.

Figure 15: FGVC-Aircraft: (a) Pairs with highest positive influence, (b) Pairs with highest negative influence.