

# What Neural Networks Memorize and Why: Discovering the Long Tail via Influence Estimation

Vitaly Feldman \*†  
Apple

Chiyuan Zhang\*  
Google Research, Brain Team

## Abstract

Deep learning algorithms are well-known to have a propensity for fitting the training data very well and often fit even outliers and mislabeled data points. Such fitting requires memorization of training data labels, a phenomenon that has attracted significant research interest but has not been given a compelling explanation so far. A recent work of Feldman [Fel19] proposes a theoretical explanation for this phenomenon based on a combination of two insights. First, natural image and data distributions are (informally) known to be long-tailed, that is have a significant fraction of rare and atypical examples. Second, in a simple theoretical model such memorization is necessary for achieving close-to-optimal generalization error when the data distribution is long-tailed. However, no direct empirical evidence for this explanation or even an approach for obtaining such evidence were given.

In this work we design experiments to test the key ideas in this theory. The experiments require estimation of the influence of each training example on the accuracy at each test example as well as memorization values of training examples. Estimating these quantities directly is computationally prohibitive but we show that closely-related *subsampled* influence and memorization values can be estimated much more efficiently. Our experiments demonstrate the significant benefits of memorization for generalization on several standard benchmarks. They also provide quantitative and visually compelling evidence for the theory put forth in [Fel19].

## 1 Introduction

Perhaps the most captivating aspect of deep learning algorithms is their ability to generalize to unseen data. The models used in deep learning are typically overparameterized, making it easy to perfectly fit the training dataset without any generalization. In fact, the standard training algorithms do fit the training data very well, typically achieving 95-100% accuracy, even when the accuracy on the test dataset is much more modest. In particular, they usually fit obvious outliers (such as images with no discernible features of their class) and mislabeled examples. The only way for a training algorithm to fit an example whose label cannot be predicted based on the rest of the dataset is to memorize<sup>1</sup> the label. Further, it is now well-known that standard deep learning algorithms achieve high training accuracy even on large and randomly labeled datasets [Zha+17].

This propensity for label memorization is not explained by the standard approach to understanding of generalization. At a high level, the standard approach upper-bounds the generalization error by the sum of an upper bound on the generalization gap controlled by a model complexity (or stability) parameter and the empirical error. Fitting of outliers and mislabeled examples does not improve the generalization error. Therefore, to avoid “overfitting”, the balance between the complexity parameter and the empirical error is supposed to be tuned in a way that prevents label memorization. Memorization is also generally thought of (and taught in ML courses) as being the opposite of generalization.

This disconnect between the classical theory and modern practice was highlighted in the work of Zhang et al. [Zha+17] and generated a large wave of research interest in the topic of generalization for deep learning. The bulk of this research focuses on finding new ways to control the generalization gap or showing that training algorithms induce

---

\*Equal contribution.

†Part of this work done while the author was at Google Research, Brain Team.

<sup>1</sup>This notion of label memorization is defined rigorously in eq. (1) (Sec. 1.1).

a form of implicit regularization. These results have lead to tighter theoretical bounds and, in some cases, bounds that show correlation with the actual generalization gap (see [Ney+17; NK19; Jia+20a] for analyses of numerous measures). Yet, fundamentally, these works still follow the same classical approach to generalization that cannot explain memorization. Another line of research focuses on the generalization error of algorithms that fit the data perfectly (referred to as interpolating) [Wyn+17; BRT18; BHM18; LR18; BMM18; RZ19; Bar+19; BHX19; Has+19; MVS19]. At a high level, these works show that under certain conditions interpolating algorithms achieve (asymptotically) optimal generalization error. However, under the same conditions there also exist standard non-interpolating algorithms that achieve the optimal generalization error (e.g. via appropriate regularization). Thus these works do not explain why interpolating algorithms are used in the first place.

A recent work of Feldman [Fel19] proposes a new explanation for the benefits of memorization. The explanation suggests that memorization is necessary for achieving close-to-optimal generalization error when the data distribution is long-tailed, namely, rare and atypical instances make up a significant fraction of the data distribution. Moreover, in such distributions useful examples from the “long tail” (in the sense that memorizing them improves the generalization error) can be statistically indistinguishable from the useless one, such as outliers and mislabeled ones. This makes memorization of useless examples (and the resulting large generalization gap) necessary for achieving close-to-optimal generalization error. We will refer to this explanation as the *long tail theory*.

In [Fel19] the need for memorization and statistical indistinguishability of useful from useless examples are theoretically demonstrated using an abstract model. In this model the data distribution is a mixture of subpopulations and the frequencies of those subpopulations are chosen from a long-tailed prior distribution. Subpopulation are presumed to be distinct enough from each other that a learning algorithm cannot achieve high accuracy on a subpopulation without observing any representatives from it. The results in [Fel19] quantify the cost of not memorizing in terms of the prior distribution and the size of the dataset  $n$ . They also show that the cost is significant for the prototypical long-tailed distributions (such as the Zipf distribution) when the number of samples is smaller than the number of subpopulations.

While it has been recognized in many contexts that modern datasets are long-tailed [ZAR14; VHP17; BS19], it is unclear whether this has any relationship to memorization by modern deep learning algorithms and (if so) how significant is the effect. The theoretical explanation in [Fel19] is based on a generative prior distribution and therefore cannot be directly verified. This leads to the question of how the long tail theory can be tested empirically.

## 1.1 Overview

In this work we develop approaches for empirically validating the long tail theory. The starting point for such validation is examining which training examples are memorized and what is the utility of all the memorized examples as a whole. To make it more concrete we recall the definition of label memorization from [Fel19]. For a training algorithm  $\mathcal{A}$  operating on a dataset  $S = ((x_1, y_1), \dots, (x_n, y_n))$  the amount of label memorization by  $\mathcal{A}$  on example  $(x_i, y_i) \in S$  is defined as

$$\text{mem}(\mathcal{A}, S, i) := \Pr_{h \leftarrow \mathcal{A}(S)}[h(x_i) = y_i] - \Pr_{h \leftarrow \mathcal{A}(S \setminus i)}[h(x_i) = y_i], \quad (1)$$

where  $S \setminus i$  denotes the dataset  $S$  with  $(x_i, y_i)$  removed and probability is taken over the randomness of the algorithm  $\mathcal{A}$  (such as random initialization). This definition captures and quantifies the intuition that an algorithm memorizes the label  $y_i$  if its prediction at  $x_i$  based on the rest of the dataset changes significantly once  $(x_i, y_i)$  is added to the dataset.

The primary issue with this definition is that estimating memorization values with standard deviation of  $\sigma$  requires running  $\mathcal{A}(S \setminus i)$  on the order of  $1/\sigma^2$  times for every example. As a result, this approach requires  $\Omega(n/\sigma^2)$  training runs which translates into millions of training runs needed to achieve  $\sigma < 0.1$  on a dataset with  $n = 50,000$  examples. We bypass this problem by proposing a closely-related estimator that looks at the expected memorization of the label of  $x_i$  on a random subset of  $S$  that includes  $m \leq n$  of examples. It can be seen as  $\text{mem}(\mathcal{A}, S, i)$  smoothed by the random subsampling of the dataset and is also related to the Shapley value of example  $i$  for accuracy on itself. Most importantly, for  $m$  bounded away from  $n$  and 1 this memorization value can be estimated with standard deviation  $\sigma$  for every  $i$  at the same time using just  $O(1/\sigma^2)$  training runs.

We compute memorization value estimates on the MNIST, CIFAR-100 and ImageNet datasets and then estimate the marginal effect of memorized examples on the test accuracy by removing those examples from the training dataset.

We find that, aside from the MNIST dataset,<sup>2</sup> a significant fraction of examples have large memorization estimates. The marginal utility of the memorized examples is also significant, in fact higher than a random set of examples of the same size. For example, on the ImageNet  $\approx 32\%$  of examples have memorization estimates  $\geq 0.3$  and their marginal utility is  $\approx 3.4\%$  (vs.  $\approx 2.6\%$  for a random subset of  $32\%$  of examples). In addition, by visually examining the memorization estimates, we see that examples with high memorization scores are a mixture of atypical examples and outliers/mislabeled examples (whereas examples with low memorization estimates are much more typical). All of these findings are consistent with the long tail theory.

A more important prediction of the theory is that memorization is necessary since each memorized representative of a rare subpopulation significantly increases the prediction accuracy on its subpopulation. We observe that this prediction implies that there should exist a substantial number of memorized training examples each of which significantly increases the accuracy on an example in the test set. Further, of the test examples that are influenced significantly, most are influenced significantly only by a single training example. The uniqueness is important since, according to the theoretical results, such unique representatives of a subpopulation are the ones that are hard to distinguish from outliers and mislabeled examples (and thus memorizing them requires also memorizing useless examples).

To find such high-influence pairs of examples we need to estimate the influence of each training example  $(x_i, y_i)$  on the accuracy of the algorithm  $\mathcal{A}$  at each test example  $(x'_j, y'_j)$ :

$$\text{infl}(\mathcal{A}, S, i, j) := \Pr_{h \leftarrow \mathcal{A}(S)} [h(x'_j) = y'_j] - \Pr_{h \leftarrow \mathcal{A}(S \setminus i)} [h(x'_j) = y'_j]. \quad (2)$$

As with memorization values, estimating the influence values for all pairs of examples is clearly not computationally feasible. A famous proxy for the classical leave-one-one influence is the influence function [CW82]. Computing this function for deep neural networks has been studied recently by Koh and Liang [KL17] who proposed a method based on assumptions of first and second order optimality. Alternative proxies for measuring influence have been studied in [Yeh+18; Pru+20].

We propose and use a new estimator for influence based on the same subsampling as our memorization value estimator. Its primary advantages are that it is a natural smoothed version of the influence value itself and, as we demonstrate visually, provides reliable and relatively easy to interpret estimates. We then locate all train-test pairs of examples from the same class  $(x_i, y_i)$  and  $(x'_j, y'_j)$  such that our estimate of the memorization value of  $(x_i, y_i)$  is sufficiently large (we chose 0.25 as the threshold) and our estimate of the influence of  $(x_i, y_i)$  on the accuracy at  $(x'_j, y'_j)$  is significant (we chose 0.15 as the threshold). See Sec. 2 for the details of the estimator and the justification of the threshold choice. Overall, we found a substantial number of such pairs in the CIFAR-100 and the ImageNet. For example we found 1641 pairs satisfying these criteria in the ImageNet. In these pairs there are 1462 different test examples (comprising 2.92% of the test set) of which 1298 are influenced significantly by only a single training example.

These quantitative results of our experiments clearly support the key ideas of the long tail theory. To further investigate the findings, we visually inspect the high-influence pairs of examples that were found by our methods. This inspection shows that, in most cases, the pairs have an easy-to-interpret visual similarity and provide, we believe, the most compelling evidence for the long tail theory.

In addition to our main experiments, we investigate several natural related questions (albeit only on CIFAR-100). In the first set of experiments we look at how much the results of our experiments depend on the choice of the architecture. We find that, while the architecture definitely plays a role (as it does for the accuracy) there are strong correlations between sets of memorized examples and high-influence pairs for different architectures. These experiments also give a sense of how much our results are affected by the randomness in the estimation and training processes and the resulting selection bias.

Finally, as our influence and memorization estimators are still very computationally intensive, we consider a faster way to compute these values. Specifically, instead of training the entire network on each random subset of  $S$ , we train only the last layer over the representation given by the penultimate layer of the network trained once on the entire dataset. The resulting estimator is much more computationally efficient but it fails completely at detecting memorized examples and gives much worse influence estimates. In addition to being a potentially useful negative result, this

---

<sup>2</sup>We include the MNIST dataset primarily as a comparison point, to show that memorization plays a much smaller role when the variability of the data is low (corresponding to a low number of subpopulations in a mixture model) and the number of examples per class is high.

experiment provides remarkable evidence that most of the memorization effectively happens in the deep representation and not in the last layer.

## 1.2 Related Work

For a more detailed comparison of the long tail theory with the standard approaches to understanding of generalization and work on interpolating methods we refer the reader to [Fel19].

Memorization of data has been investigated in the context of privacy-preserving ML. Starting from [Sho+17], multiple works have demonstrated that the output of the trained neural network can be used to perform successful membership inference attacks, that is to infer with high accuracy whether a given data point is part of the training set. An important problem in this area is to find learning algorithms that are more resistant to such attacks on privacy. Our results suggest that reducing memorization will also affect the accuracy of learning.

Arpit et al. [Arp+17] examine the relationship between memorization of random labels and performance of the network on true labels. The work demonstrates that using various regularization techniques, it is possible to reduce the ability of the training algorithm to fit random labels without significantly impacting its test accuracy on true labels. The explanation proposed for this finding is that memorization is not necessary for learning. However memorization is used informally to refer to fitting the entire randomly labeled dataset. Even with regularization, the algorithms used in this work memorize a significant fraction of randomly labeled examples and fit the true training data (nearly) perfectly.

Carlini et al. [CEP19] consider different ways to measure how “prototypical” each of the data points is according to several metrics and across multiple datasets. They examine 5 different metrics and draw a variety of insights about the metrics from the visualisations of rankings along these metrics. They also discuss “memorized exceptions”, “uncommon submodes” and outliers informally. While the memorization value we investigate also identifies atypical examples and outliers, it is not directly related to these metrics. This work also briefly reports on an unsuccessful attempt to find individual training examples that influence individual test examples on MNIST. As our experiments demonstrate, such high-influence pairs are present in MNIST and therefore this result confirms that finding them via the direct leave-one-out method while ensuring statistical significance is computationally infeasible even on MNIST.

The use of random data subsamples is standard in data analysis, most notably in bagging, bootstrapping and cross validation. In these applications the results from random subsamples are aggregated to estimate the properties of the results of data analysis as a whole whereas we focus on the properties of individual samples. Concurrent work of Jiang et al. [Jia+20b] uses data subsamples to estimate the regularity (or easiness) of each training example. This score (referred to as the *empirical consistency score*) coincides with the second term in our subsampled memorization value estimator (Alg. 1, line 5). The value of this estimator is typically equal to one minus our memorization estimate since fitting hard-to-predict training examples requires memorizing their labels. The score was derived independently of our work and its computation in [Jia+20b] builds on the experimental framework developed in this work. The focus in [Jia+20b] is on the investigation of several proxies for the consistency score and their experimental results are otherwise unrelated to ours.

Toneva et al. [Ton+19] investigate a “forgetting” phenomenon in which an example that was predicted correctly at some point during training becomes misclassified. They empirically demonstrate that examples that are never “forgotten” tend to have low marginal utility. The notion of memorization we consider is not directly related to such “forgetting”.

## 2 Estimation and Selection Procedures

In this section we describe how our memorization and influence estimators are defined and computed. We also describe the selection criteria for high-influence pairs of examples.

**Memorization and influence estimators:** Our goal is to measure the label memorization by an algorithm  $\mathcal{A}$  on a (training) dataset  $S$  and example  $(x_i, y_i)$  (eq. (1)) and the influence of a training example  $(x_i, y_i)$  on a test example  $(x'_j, y'_j)$ . Both of these values are a special case of measuring the influence of  $(x_i, y_i)$  on the expected accuracy at some example  $z = (x, y)$  or

$$\text{inf1}(\mathcal{A}, S, i, z) := \Pr_{h \leftarrow \mathcal{A}(S)}[h(x) = y] - \Pr_{h \leftarrow \mathcal{A}(S \setminus i)}[h(x) = y]. \quad (3)$$

Clearly,  $\text{mem}(\mathcal{A}, S, i) = \text{inf1}(\mathcal{A}, S, i, (x_i, y_i))$ , that is memorization corresponds to the influence of example  $i$  on the accuracy on itself (or self-influence).

As discussed in Sec. 1.1, directly estimating the influence of all  $n$  training examples within standard deviation  $\sigma$  requires training on the order of  $n/\sigma^2$  models. Thus we propose a closely-related influence value that looks at the expected influence of an example  $(x_i, y_i)$  relative to a dataset that includes a random subset of  $S$  of size  $m < n$ . More formally, for a set of indices  $I \subseteq [n]$  ( $[n]$  is defined as the set  $\{1, \dots, n\}$ ), let  $S_I = (x_i, y_i)_{i \in I}$  be the dataset consisting of examples from  $S$  with indices in  $I$ . For a set of indices  $J \subseteq [n]$ , let  $P(J, m)$  denote the uniform distribution over all subsets of  $J$  of size  $m$ . Then we define:

$$\text{inf1}_m(\mathcal{A}, S, i, z) := \mathbf{E}_{I \sim P([n] \setminus \{i\}, m-1)} [\text{inf1}(\mathcal{A}, S_{I \cup \{i\}}, i, z)],$$

where by sampling a random subset of size  $m-1$  that excludes index  $i$  we ensure that  $S_{I \cup \{i\}}$  is uniform over all subsets of size  $m$  that include the index  $i$ .

We now show that subsampled influence values can be estimated with standard deviation  $\sigma$  by training just  $O(1/\sigma^2)$  models.

**Lemma 2.1.** *There exists an algorithm that for every dataset  $S \in (X \times Y)^n$ , learning algorithm  $\mathcal{A}$ ,  $m \in [n]$  and integer  $t$ , runs  $\mathcal{A}$   $t$  times and outputs estimates  $(\mu_i)_{i \in [n]}$  such that for every  $i \in [n]$  and  $p = \min(m/n, 1 - m/n)$ ,*

$$\mathbf{E}[(\text{inf1}_m(\mathcal{A}, S, i, z) - \mu_i)^2] \leq \frac{1}{pt} + \frac{1}{(1-p)t} + \frac{e^{-pt/16}}{2},$$

where the expectation is with respect to the randomness of  $\mathcal{A}$  and the randomness of the estimation algorithm.

We include the proof in Sec. A. The estimator exploits the fact that by training models on random subsets of size  $m$  we will, with high probability, obtain many models trained on subsets that include index  $i$  for every  $i$  and also many subsets that exclude  $i$ . By linearity of expectation, this gives an estimate of  $\text{inf1}_m(\mathcal{A}, S, i, z)$ . Alg. 1 describes the resulting algorithm for estimating all the memorization and influence values. We use  $k \sim [t]$  to denote index  $k$  being chosen randomly and uniformly from the set  $[t]$  (the probabilities for such sampling are computed by enumerating over all values of  $k$ ).

---

#### Algorithm 1 Memorization and influence value estimators

---

**Require:** Training dataset:  $S = ((x_1, y_1), \dots, (x_n, y_n))$ , testing dataset  $S_{test} = ((x'_1, y'_1), \dots, (x'_{n'}, y'_{n'}))$ , learning algorithm  $\mathcal{A}$ , subset size  $m$ , number of trials  $t$ .

- 1: Sample  $t$  random subsets of  $[n]$  of size  $m$ :  $I_1, I_2, \dots, I_t$ .
  - 2: **for**  $k = 1$  to  $t$  **do**
  - 3:   Train model  $h_k$  by running  $\mathcal{A}$  on  $S_{T_k}$ .
  - 4:   **for**  $i = 1$  to  $n$  **do**
  - 5:      $\widetilde{\text{mem}}_m(\mathcal{A}, S, i) := \Pr_{k \sim [t]}[h_k(x_i) = y_i \mid i \in I_k] - \Pr_{k \sim [t]}[h_k(x_i) = y_i \mid i \notin I_k]$ .
  - 6:     **for**  $j = 1$  to  $n'$  **do**
  - 7:        $\widetilde{\text{inf1}}_m(\mathcal{A}, S, i, j) := \Pr_{k \sim [t]}[h_k(x'_j) = y'_j \mid i \in I_k] - \Pr_{k \sim [t]}[h_k(x'_j) = y'_j \mid i \notin I_k]$ .
  - 8: **return**  $\widetilde{\text{mem}}_m(\mathcal{A}, S, i)$  for all  $i \in [n]$ ;  $\widetilde{\text{inf1}}_m(\mathcal{A}, S, i, j)$  for all  $i \in [n], j \in [n']$ .
- 

The larger the subset size parameter  $m$ , the closer is our estimator to the original  $\text{inf1}(\mathcal{A}, S, i, z)$ . At the same time, we need to ensure that we have a sufficient number of random datasets that exclude each example  $(x_i, y_i)$ . To roughly balance these requirements in all of our experiments we chose  $m = 0.7 \cdot n$ . Due to computational constraints we chose the number of trials to be  $t = 2000$  for ImageNet and  $t = 4000$  for MNIST/CIFAR-100.

We remark that for  $m = n/2$  our influence value is closely-related to the Shapley value of example  $(x_i, y_i)$  for the function that measures the expected accuracy of the model on point  $z$ . This follows from the fact that for functions that

are symmetric (do not depend on the order of examples) the Shapley value is equal to the expected marginal utility of example  $i$  relative to the random and uniform subset of all examples. For sufficiently large  $n$ , such subsets have size close to  $n/2$  with high probability. The Shapley value is itself a natural and well-studied measure of the contribution of each point to the value of a function and thus provides an additional justification for the use of our subsampled influence values.

**Selection of high-influence pairs of examples:** To find the high-influence pairs of examples we select all pairs of examples  $(x_i, y_i) \in S$  and  $(x'_j, y'_j) \in S_{test}$  for which  $\widehat{\text{mem}}_m(\mathcal{A}, S, i) \geq \theta_{\text{mem}}$ ;  $\widehat{\text{inf1}}_m(\mathcal{A}, S, i, j) \geq \theta_{\text{inf1}}$  and  $y_i = y'_j$ . The last condition is used since the long tail theory only explains improvement in accuracy from examples in the same subpopulation and allows to reduce the noise in the selected estimates.

Selection of pairs that is based on random estimates introduces selection bias into the estimates of values that are close to the threshold value. The choice of  $\theta_{\text{mem}}$  is less important and is almost unaffected by bias due to a relatively small set of estimates. We have chosen  $\theta_{\text{mem}} = 0.25$  as a significant level of memorization. In choosing  $\theta_{\text{inf1}}$  we wanted to ensure that the effect of this selection bias is relatively small. To measure this effect we ran our selection procedure with various thresholds on CIFAR-100 twice, each based on 2000 trials. We chose  $\theta_{\text{inf1}} = 0.15$  as a value for which the Jaccard similarity coefficient between the two selected sets is  $\geq 0.7$ . In these two runs 1095 and 1062 pairs were selected, respectively with  $\approx 82\%$  of the pairs in each set also appearing in the other set. We have used the same thresholds for all three datasets for consistency. More details on the consistency of our selection process can be found in Sec. 3.5.

## 3 Empirical Results

In this section we describe the results of the experiments based on the methods and parameters specified in Sec. 2. We use ResNet50 in both ImageNet and CIFAR-100 experiments, which is a Residual Network architecture widely used in the computer vision community [He+16]. Full details of the experimental setup and training algorithms are given in Sec. B.

### 3.1 Examples of memorization value estimates

In Fig. 1 we show examples of the estimated subsampled memorization values around 0, 0.5, and 1, respectively. Additional examples can be found at [FZ20]. These examples suggest that the estimates reflect our intuitive interpretation of label memorization. In particular, some of the examples with estimated memorization value close to 0 are clearly typical whereas those with value close to 1 are atypical, highly ambiguous or mislabeled.

### 3.2 Marginal utility of memorized examples

Fig. 2 demonstrates the significant effect that the removal of memorized examples from the dataset has on the test set accuracy. One could ask whether this effect is solely due to the reduction in number of available training examples as a result of the removal. To answer this question we include in the comparison the accuracy of the models trained on the identical number of examples which are chosen randomly from the entire dataset. Remarkably, memorized examples have higher marginal utility than the identical number of randomly chosen examples. The likely reason for this is that most of the randomly chosen examples are easy and have no marginal utility.

### 3.3 Estimation of influence and marginal utility of high-influence examples

We compute the estimated influences and select high-influence pairs of examples as described in Sec. 2. Overall we found 35/1015/1641 pairs in MNIST/CIFAR-100/ImageNet. In Fig. 3 we give histograms of the number of such pairs for every level of influence. The number of unique test examples in these pairs is 33/888/1462 (comprising 0.33%/8.88%/2.92% of the test set). Of those 31/774/1298 are influenced (above the 0.15 threshold) by a single training example. On CIFAR-100 and ImageNet this confirms the importance of the subpopulations in the long tail that have unique representatives for the generalization error. The results on the MNIST are consistent with the fact that it is a

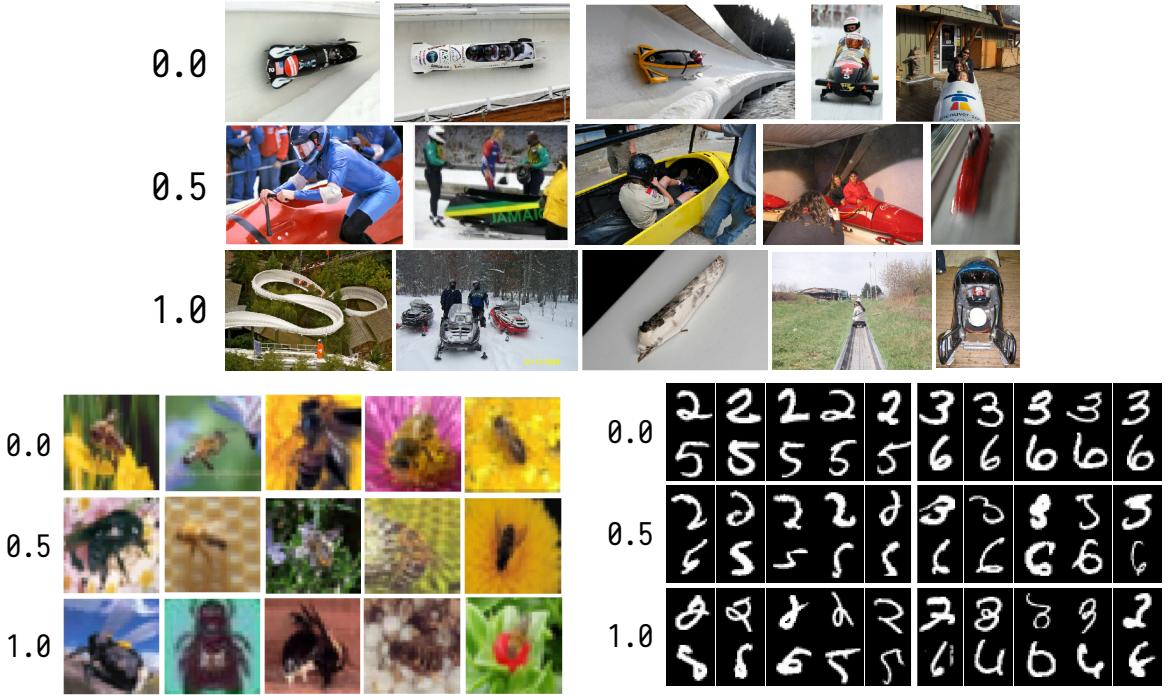


Figure 1: Examples of memorization values from ImageNet class “bobsled” (top), CIFAR-100 class “bee” (bottom left) and MNIST class 2, 3, 5, 6 (bottom right).

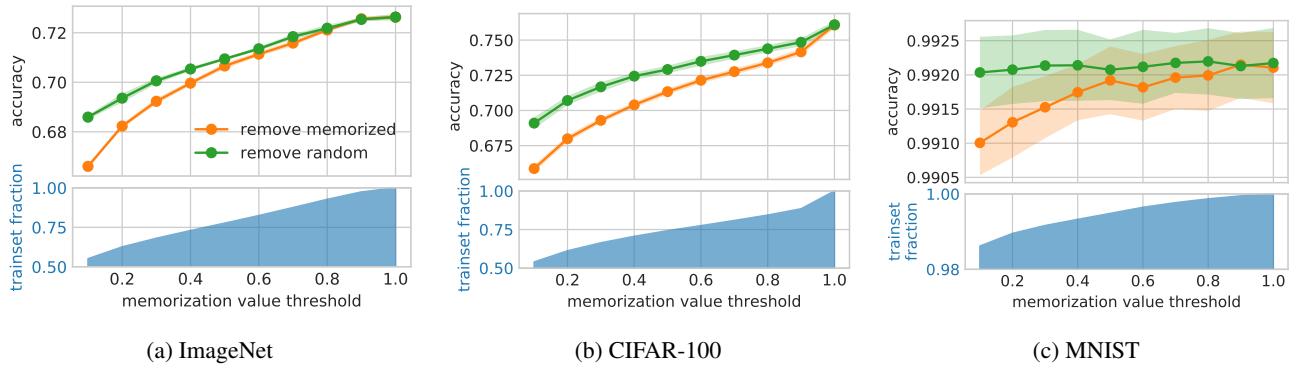


Figure 2: Effect on the test set accuracy of removing examples with memorization value estimate above a given threshold and the same number of randomly chosen examples. Fraction of the training set remaining after the removal is in the bottom plots. Shaded area in the accuracy represents one standard deviation on 100 (CIFAR-100, MNIST) and 5 (ImageNet) trials.

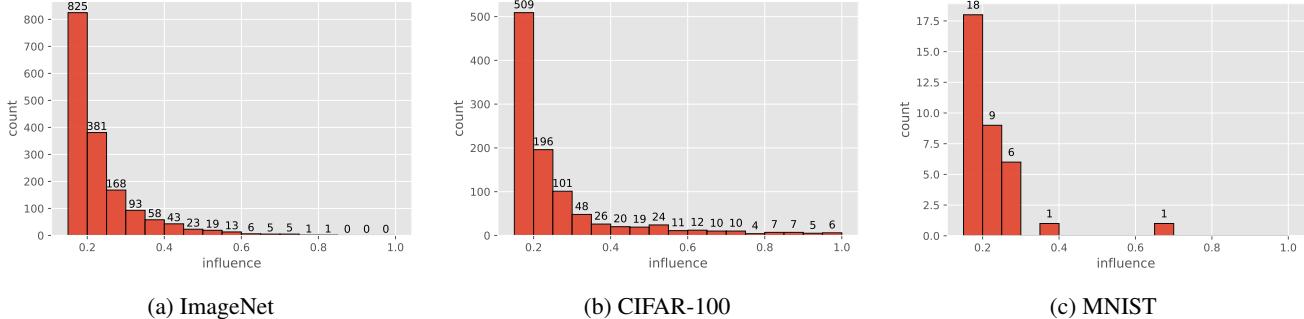


Figure 3: Histogram of the influence estimates for all the pairs from the ImageNet, CIFAR-100 and MNIST datasets that were selected according to Algorithm 1 and criteria described in Sec. 2.

much easier dataset with low variation among the examples (in particular, a smaller number of subpopulations) and much larger number of examples per class.

Next we examine the marginal utility of the training examples in high-influence pairs. Denote the set of all the unique training and testing examples in the high-influence pairs by  $S_h \subseteq S$  and  $S'_h \subseteq S'$ , respectively. To evaluate the marginal utility of the high-influence training examples  $S_h$ , we train a ResNet50 model on the CIFAR-100 full training set  $S$ , and on  $S \setminus S_h$ , respectively. Over 100 random runs, the two settings result in  $76.06 \pm 0.28\%$  and  $73.52 \pm 0.25\%$  accuracy on the test set, respectively, giving  $2.54 \pm 0.2\%$  difference. When restricted to the set of highly-influenced test examples  $S'_h$ , the two settings have  $72.14 \pm 1.32\%$  and  $45.38 \pm 1.45\%$  accuracy, respectively. Note that the difference in accuracy on these examples contributes  $2.38 \pm 0.17\%$  to the total difference in accuracy. This difference is within one standard deviation of the entire difference in accuracy which means that the high influences that we detected capture the marginal utility of  $S_h$ . This shows that there is a large number of memorized training examples for which the only benefit of memorization is the large increase in accuracy on individual test examples. This is well aligned with modeling used in the long tail theory where the label memorization on representatives of rare subpopulations significantly increases the accuracy on those subpopulations [Fel19].

### 3.4 Examples of high-influence pairs

Remarkably, in most cases our estimates of influence are easy to interpret by humans. Very high influence scores (greater than 0.4) almost always correspond to near duplicates or images from a set of photos taken together. These are artifacts of the data collection methods and are particularly prominent in CIFAR-100 which has numerous near-duplicate images [BD20]. Naturally, such examples benefit the most from memorization. Examples in pairs with lower influences (more than 80% of influences we found are below 0.4) are visually very similar but in most cases are not from the same set. We include examples of various influences for MNIST, CIFAR-100 and ImageNet in Figs. 4, 5 and 6, respectively. Additional examples can be found at [FZ20]. To select the presented examples, we first sort the training examples in the high-influence pairs by the highest influence they have on a test example and then pick 3 consecutive sets each of 5 training examples with indices spread evenly in the sorted order (in particular, without any cherry-picking, see Sec.C for the exact description).

### 3.5 Estimation consistency and comparison of different architectures

We study the consistency of our estimation of memorization and influence under subset resampling (and randomness of the training algorithm) and also across different neural network architectures on CIFAR-100. All estimates are based on 2000 trials. To measure the consistency we consider Jaccard similarity coefficient of the sets of examples that have memorization/influence estimate above a certain threshold and also average difference in estimates of examples in these sets. In particular, for memorization estimates  $\widehat{\text{mem}}_m(\mathcal{A}, S, i)$  and  $\widehat{\text{mem}}_m(\mathcal{A}', S, i)$ , where  $\mathcal{A}$  and  $\mathcal{A}'$  are training algorithms using two neural network architectures, we compare the estimates at each threshold  $\theta_{\text{mem}}$  in the following

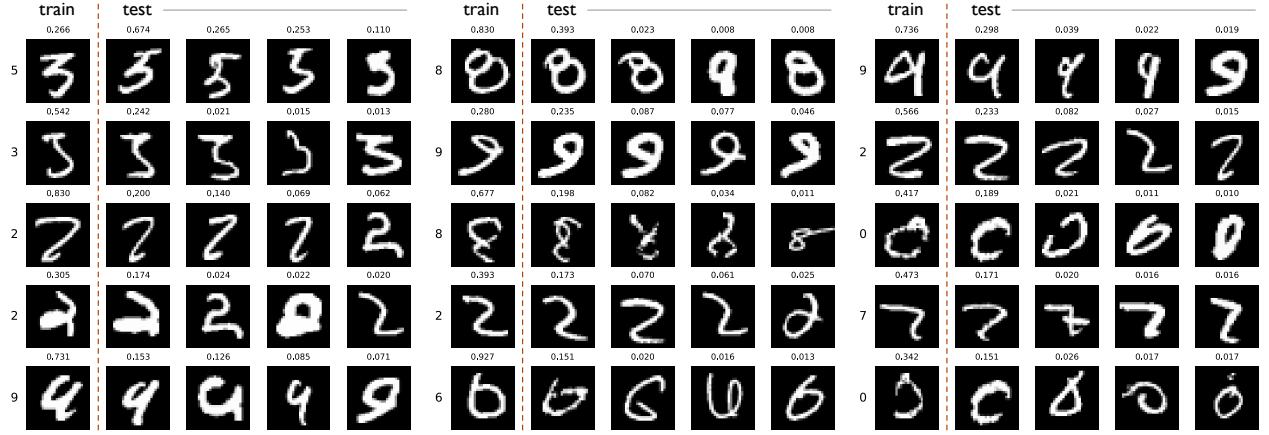


Figure 4: Examples of influence estimates for memorized examples from MNIST. Left column is the memorized examples in the training set and their memorization estimates (above). For each training example 4 most influenced examples in the test set are given together with the influence estimates (above each image).

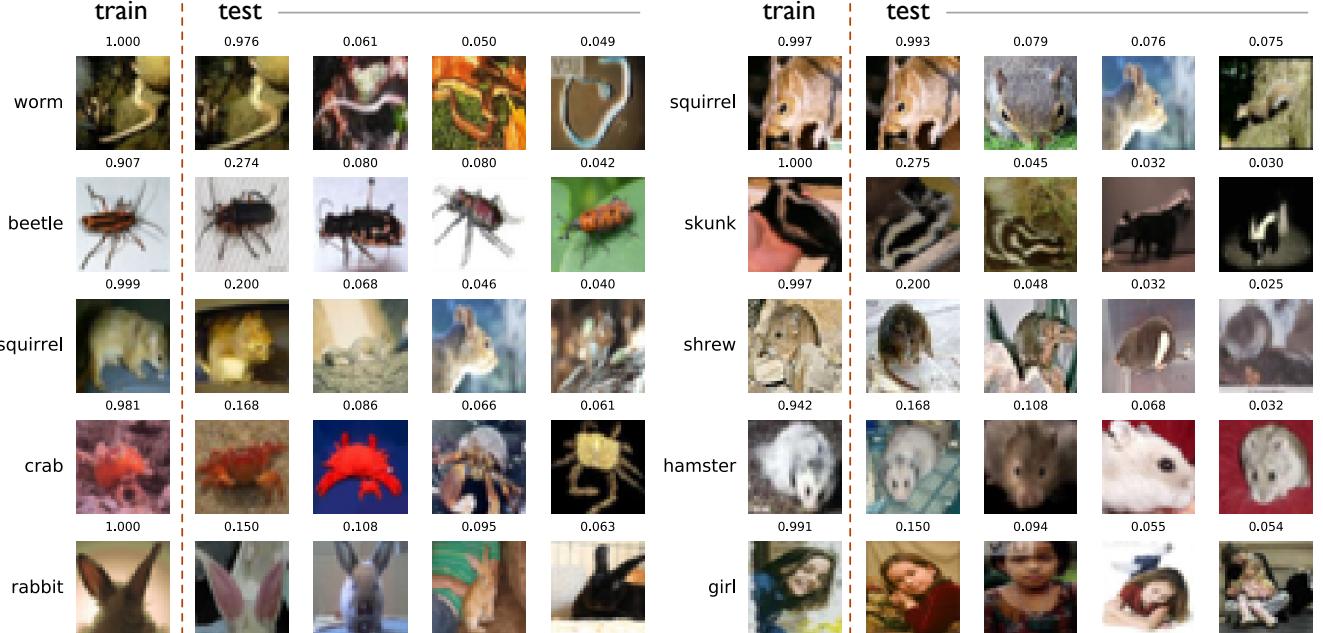


Figure 5: Additional examples of influence estimates for memorized examples from CIFAR-100. Format is as in Fig.4.

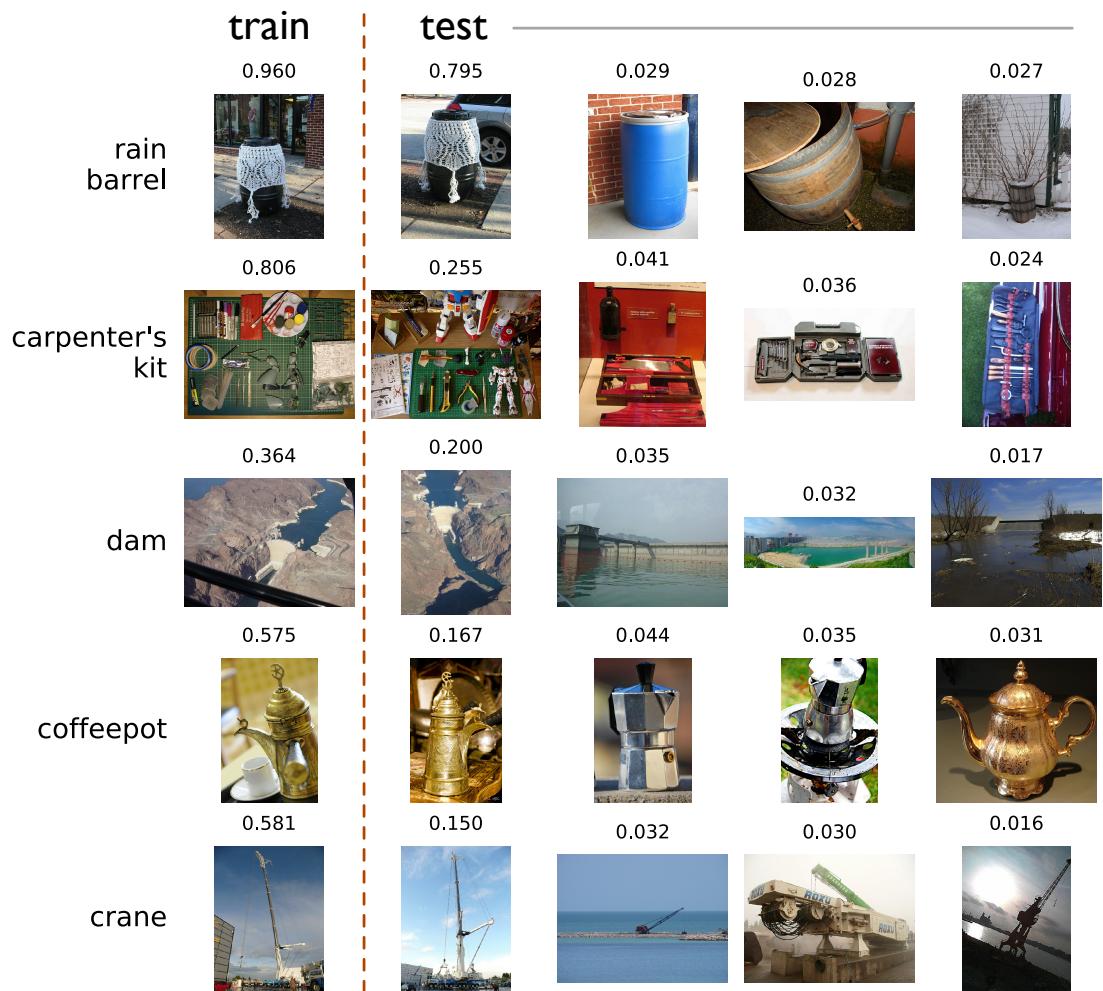


Figure 6: Additional examples of influence estimates for memorized examples from ImageNet. Format is as in Fig.4.

way: let  $I_{\text{mem}}(\theta_{\text{mem}}) := \{i : \widetilde{\text{mem}}_m(\mathcal{A}, S, i) \geq \theta_{\text{mem}}\}$ ,  $I'_{\text{mem}}(\theta_{\text{mem}}) := \{i : \widetilde{\text{mem}}_m(\mathcal{A}', S, i) \geq \theta_{\text{mem}}\}$ , then

$$D_{\text{mem}}(\theta_{\text{mem}}) := \text{mean}_{i \in I_{\text{mem}}(\theta_{\text{mem}}) \cup I'_{\text{mem}}(\theta_{\text{mem}})} |\widetilde{\text{mem}}_m(\mathcal{A}, S, i) - \widetilde{\text{mem}}_m(\mathcal{A}', S, i)| \quad (4)$$

measures the difference in memorization estimates between  $\mathcal{A}$  and  $\mathcal{A}'$  at  $\theta_{\text{mem}}$ . Similarly, the discrepancy for influence estimation at  $\theta_{\text{infl}}$  is measured by comparing  $\widetilde{\text{infl}}_m(\mathcal{A}, S, i)$  and  $\widetilde{\text{infl}}_m(\mathcal{A}', S, i)$  over the union of the two subsets:

$$I_{\text{infl}}(\theta_{\text{infl}}) := \{(i, j) : \widetilde{\text{infl}}_m(\mathcal{A}, S, i, j) \geq \theta_{\text{infl}}, \widetilde{\text{mem}}_m(\mathcal{A}, S, i) \geq 0.25\} \text{ and}$$

$$I'_{\text{infl}}(\theta_{\text{infl}}) := \{(i, j) : \widetilde{\text{infl}}_m(\mathcal{A}', S, i, j) \geq \theta_{\text{infl}}, \widetilde{\text{mem}}_m(\mathcal{A}', S, i) \geq 0.25\}.$$

Note we have an extra constraints of memorization estimate being above 0.25, which is used when we select high-influence pairs.

Jaccard similarity coefficient for these sets is defined as:

$$J_{\text{mem}}(\theta_{\text{mem}}) := \frac{|I_{\text{mem}}(\theta_{\text{mem}}) \cap I'_{\text{mem}}(\theta_{\text{mem}})|}{|I_{\text{mem}}(\theta_{\text{mem}}) \cup I'_{\text{mem}}(\theta_{\text{mem}})|}$$

and similarly,

$$J_{\text{infl}}(\theta_{\text{infl}}) := \frac{|I_{\text{infl}}(\theta_{\text{infl}}) \cap I'_{\text{infl}}(\theta_{\text{infl}})|}{|I_{\text{infl}}(\theta_{\text{infl}}) \cup I'_{\text{infl}}(\theta_{\text{infl}})|}.$$

We compare ResNet50 with ResNet50 (independent runs with the same architecture), ResNet18, Inception [Sze+15], and DenseNet100 [Hua+17] in Fig. 7. The results show consistency in the estimation of both memorization and influence across different architectures.

We first note that comparison of two different runs of the same architecture gives a sense of the accuracy of our estimates and the effect of selection bias. For memorization, the high consistency and almost non-existent selection bias are apparent. (Jaccard similarity is not very reliable when sets are relatively small and, as a result, we see a drop near threshold 1.0 even though there is almost no difference in the estimates). For influence estimates there is a clear drop in accuracy and consistency around threshold 0.1 which appears to be primarily due to the selection bias. At the same time, the average difference in estimates is still below 0.015. Our choice of influence threshold being 0.15 was in part guided by ensuring that Jaccard similarity for the chosen threshold is sufficiently high (above 0.7).

The plots also show high similarity between memorized examples and high-influence pairs computed for the different architectures. The difference in the produced estimates appears to be closely correlated with the difference in the accuracy of the architectures. This is expected since both memorization and influence estimates rely directly on the accuracy of the models. This suggests that our memorization estimates may not be very sensitive to variations in the architectures as long as they achieve similar accuracy.

### 3.6 Does the last layer suffice for memorization?

Finally, we explore a natural approach to speed up the computation of our estimator. We train a ResNet50 model on the full CIFAR-100 training set and take the output of the penultimate layer as the *representation* for each example. Then, when training with subsets of size  $m$ , we start from the pre-trained *representations* and only learn a fresh linear classifier on top of that from a random initialization. This reduces the training time by a factor of 720. The intuition is that, if label memorization mostly happens at the final layer, then we could derive similar influence estimates much faster. In principle, this could be true, as the final layer in many classification networks is itself overparameterized (e.g. the final layer of ResNet50 for CIFAR-100 has more than 200k parameters).

Our experimental results suggest that this intuition is wrong. Specifically, the 4,000 linear models trained using 70% training data achieve  $75.8 \pm 0.1\%$  accuracy on the test set. In comparison, the ResNet50 model trained on the full training set, which is used to generate the representations, achieves 75.9% test accuracy, and the 4,000 ResNet50 models trained on 70% training data achieve only  $72.3 \pm 0.3\%$  test accuracy. Moreover, there are only 38 training examples with memorization estimates above 0.25 using linear models, compared with 18,099 examples using full ResNet50 models. This suggests that most of the memorization is already present in the representation before reaching the final layer. Namely, trained representations of memorized examples are close to those of other examples from the

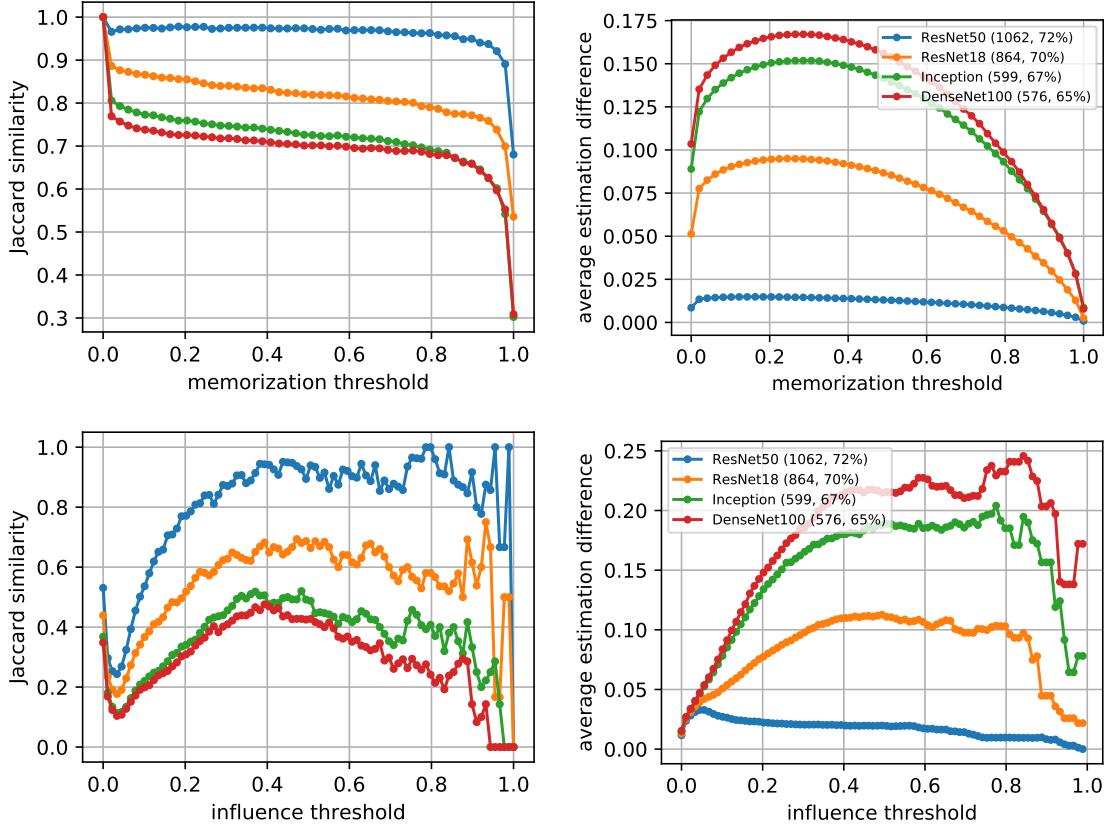


Figure 7: Consistency of the estimation of memorization (top) and influence (bottom) across different architectures on CIFAR-100. In the average estimation difference we plot  $D_{\text{mem}}(\theta_{\text{mem}})$  and  $D_{\text{infl}}(\theta_{\text{infl}})$ . Jaccard similarity plots are for  $J_{\text{mem}}(\theta_{\text{mem}})$  and  $J_{\text{infl}}(\theta_{\text{infl}})$ . All the architectures are compared to ResNet50 — with the “ResNet50” entry being comparison between two independent runs of the same architecture. The numbers in the legend indicate the number of high-influence pairs selected by each architecture according to  $\theta_{\text{infl}} = 0.15$  and  $\theta_{\text{mem}} = 0.25$ , and the average test accuracy (with 70% training set), respectively.

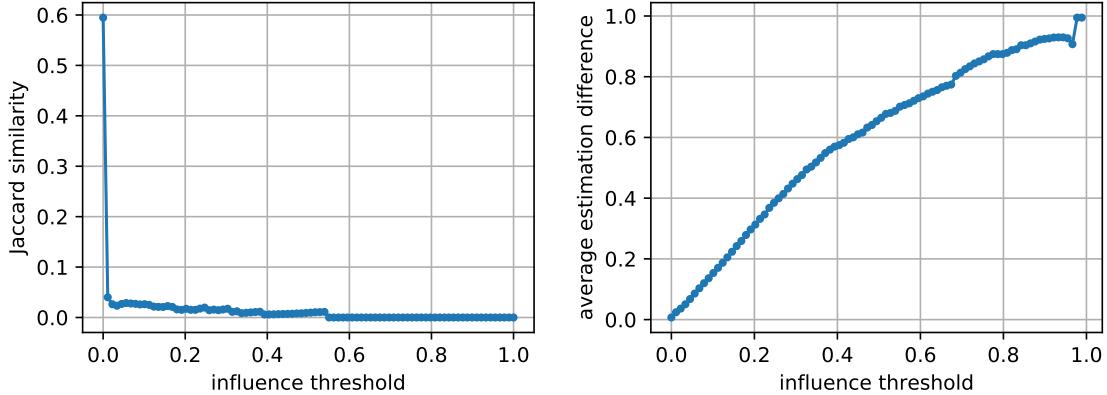


Figure 8: Consistency of the estimation of influence between ResNet50 and linear models trained on the penultimate layer representations computed on the entire CIFAR-100 dataset.

same class. Despite the lack of memorization, we still found 457 high-influence pairs of examples (as before, those with influence estimates above 0.15). In most of these pairs we see no visual similarity (although there is still a significant fraction of those that are visually similar). In Fig. 8 we quantify the large discrepancy in estimates obtained using this technique and training of the full ResNet50 model. Specifically, we compare the influence estimates using the Jaccard similarity and average difference in estimate but without the additional constraint of having memorization value above 0.25 (since it is satisfied only by 38 examples).

## 4 Discussion

Our experiments provide the first empirical investigation of memorization and its effects on accuracy that are based on formally defined and intuitive criteria. The results reveal that, in addition to outliers and mislabeled examples, neural networks memorize training examples that significantly improve accuracy on visually similar test examples. These pairs of examples are visually atypical and most train and test examples only appear in a single pair. This, we believe, strongly supports the long tail theory and, together with the results in [Fel19], provides the first rigorous and compelling explanation for the propensity of deep learning algorithms to memorize seemingly useless labels: it is a result of (implicit) tuning of the algorithms for the highest accuracy on long-tailed and mostly noiseless data.

Our work demonstrates that accuracy of a learning algorithm on long tailed data distributions depends on its ability to memorize the labels. As can be seen from the results, the effect on the accuracy of not memorizing examples depends on the number of available examples and the data variability (a formal analysis of this dependence can be found in [Fel19]). This means that the effect on accuracy will be higher on an under-represented subpopulation. The immediate implication is that techniques that limit the ability of a learning system to memorize will have a disproportionate effect on under-represented subpopulations. Techniques aimed at optimizing the model size (e.g. model compression) or training time are likely to affect the ability of the learning algorithm to memorize data. This effect is already known in the context of differential privacy (which formally limits the ability of an algorithm to memorize data) [BPS19].

The experiments in Section 3.6 demonstrate that most of memorization happens in the representations derived by training a DNN. A natural direction for future work is to derive a detailed understanding of the process of memorization by a training algorithm.

The primary technical contribution of our work is the development of influence and memorization estimators that are simple to implement, computationally feasible, and essentially as accurate as true leave-one-out influences. While several other approaches for influence estimation exist, we believe that our approach provides substantially easier to interpret results. Unlike some of the existing techniques [KL17; Yeh+18; Pru+20] it is also completely model-agnostic and is itself easy to explain. In addition to understanding of deep learning, influence estimation is useful for interpretability and outlier detection and, we hope, our estimator will find applications in these areas.

Computing our estimator with high accuracy relies on training thousands of models and thus requires significant computational resources. A natural direction for future work is finding proxies for our estimator that can be computed more efficiently. To simplify future research in this direction and other applications of our estimator, we provide the computed values for CIFAR-100 and ImageNet at [FZ20].

## References

- [Aba+15] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <https://www.tensorflow.org/>.
- [Arp+17] D. Arpit, S. Jastrzkebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, et al. “A closer look at memorization in deep networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 233–242.
- [Bar+19] P. L. Bartlett, P. M. Long, G. Lugosi, and A. Tsigler. “Benign Overfitting in Linear Regression”. In: *arXiv preprint arXiv:1906.11300* (2019).
- [BD20] B. Barz and J. Denzler. “Do We Train on Test Data? Purging CIFAR of Near-Duplicates”. In: *Journal of Imaging* 6.6 (2020), p. 41.
- [BHM18] M. Belkin, D. J. Hsu, and P. Mitra. “Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 2300–2311.
- [BHX19] M. Belkin, D. Hsu, and J. Xu. “Two models of double descent for weak features”. In: *arXiv preprint arXiv:1903.07571* (2019).
- [BMM18] M. Belkin, S. Ma, and S. Mandal. “To Understand Deep Learning We Need to Understand Kernel Learning”. In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 541–549. URL: <http://proceedings.mlr.press/v80/belkin18a.html>.
- [BPS19] E. Bagdasaryan, O. Poursaeed, and V. Shmatikov. “Differential privacy has disparate impact on model accuracy”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 15453–15462.
- [BRT18] M. Belkin, A. Rakhlin, and A. B. Tsybakov. “Does data interpolation contradict statistical optimality?” In: *arXiv preprint arXiv:1806.09471* (2018).
- [BS19] R. Babbar and B. Schölkopf. “Data scarcity, robustness and extreme multi-label classification”. In: *Machine Learning* (2019).
- [CEP19] N. Carlini, Ú. Erlingsson, and N. Papernot. “Distribution Density, Tails, and Outliers in Machine Learning: Metrics and Applications”. In: *arXiv preprint arXiv:1910.13427* (2019).
- [CW82] R. D. Cook and S. Weisberg. *Residuals and influence in regression*. New York: Chapman and Hall, 1982.
- [Fel19] V. Feldman. “Does Learning Require Memorization? A Short Tale about a Long Tail”. In: *CoRR* abs/1906.05271 (2019). Extended abstract in STOC 2020. arXiv: 1906.05271. URL: <http://arxiv.org/abs/1906.05271>.
- [FZ20] V. Feldman and C. Zhang. *Additional Material*. <https://pluskid.github.io/influence-memorization/>. 2020.
- [Has+19] T. Hastie, A. Montanari, S. Rosset, and R. J. Tibshirani. “Surprises in High-Dimensional Ridgeless Least Squares Interpolation”. In: *arXiv preprint arXiv:1903.08560* (2019).
- [He+16] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.
- [Hua+17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. “Densely connected convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.
- [Jia+20a] Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio. “Fantastic Generalization Measures and Where to Find Them”. In: *ICLR*. 2020. URL: <https://openreview.net/forum?id=SJgIPJBvH>.

- [Jia+20b] Z. Jiang, C. Zhang, K. Talwar, and M. C. Mozer. “Characterizing Structural Regularities of Labeled Data in Overparameterized Models”. In: *CoRR* abs/2002.03206 (2020). arXiv: 2002.03206. URL: <https://arxiv.org/abs/2002.03206>.
- [KL17] P. W. Koh and P. Liang. “Understanding black-box predictions via influence functions”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1885–1894.
- [LR18] T. Liang and A. Rakhlin. “Just interpolate: Kernel” ridgeless” regression can generalize”. In: *arXiv preprint arXiv:1808.00387* (2018).
- [MVS19] V. Muthukumar, K. Vodrahalli, and A. Sahai. “Harmless interpolation of noisy data in regression”. In: *arXiv preprint arXiv:1903.09139* (2019).
- [Ney+17] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. “Exploring generalization in deep learning”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5947–5956.
- [NK19] V. Nagarajan and J. Z. Kolter. “Uniform convergence may be unable to explain generalization in deep learning”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 11611–11622.
- [Pru+20] G. Pruthi, F. Liu, M. Sundararajan, and S. Kale. *Estimating Training Data Influence by Tracking Gradient Descent*. 2020. arXiv: 2002.08484 [cs.LG].
- [RZ19] A. Rakhlin and X. Zhai. “Consistency of Interpolation with Laplace Kernels is a High-Dimensional Phenomenon”. In: *COLT*. Vol. 99. PMLR, 2019, pp. 2595–2623. URL: <http://proceedings.mlr.press/v99/rakhlin19a.html>.
- [Sho+17] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. “Membership Inference Attacks Against Machine Learning Models”. In: *2017 IEEE Symposium on Security and Privacy, SP 2017*. 2017, pp. 3–18.
- [Sze+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.
- [Ton+19] 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: *ICLR*. OpenReview.net, 2019. URL: <https://openreview.net/forum?id=BJ1xm30cKm>.
- [VHP17] G. Van Horn and P. Perona. “The devil is in the tails: Fine-grained classification in the wild”. In: *arXiv preprint arXiv:1709.01450* (2017).
- [Wyn+17] A. J. Wyner, M. Olson, J. Bleich, and D. Mease. “Explaining the success of adaboost and random forests as interpolating classifiers”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 1558–1590.
- [Yeh+18] C.-K. Yeh, J. Kim, I. E.-H. Yen, and P. K. Ravikumar. “Representer point selection for explaining deep neural networks”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9291–9301.
- [ZAR14] X. Zhu, D. Anguelov, and D. Ramanan. “Capturing long-tail distributions of object subcategories”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 915–922.
- [Zha+17] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. “Understanding deep learning requires rethinking generalization”. In: *ICLR*. 2017. URL: <https://openreview.net/forum?id=Sy8gdB9xx>.

## A Proof of Lemma 2.1

*Proof.* Observe that, by linearity of expectation, we have:

$$\begin{aligned} \text{infl}_m(\mathcal{A}, S, i, z) &= \mathbf{E}_{I \sim P([n] \setminus \{i\}, m-1)} [\text{infl}(\mathcal{A}, S_{I \cup \{i\}}, i, z)] \\ &= \mathbf{E}_{I \sim P([n] \setminus \{i\}, m-1)} \left[ \Pr_{h \leftarrow \mathcal{A}(S_{I \cup \{i\}})} [h(x) = y] - \Pr_{h \leftarrow \mathcal{A}(S_I)} [h(x) = y] \right] \\ &= \Pr_{I \sim P([n] \setminus \{i\}, m-1), h \leftarrow \mathcal{A}(S_{I \cup \{i\}})} [h(x) = y] - \Pr_{I \sim P([n] \setminus \{i\}, m-1), h \leftarrow \mathcal{A}(S_I)} [h(x) = y] \end{aligned}$$

By definition, the distribution of  $I \cup \{i\}$  for  $I$  sampled from  $P([n] \setminus \{i\}, m-1)$  is the same as the distribution of  $J$  sampled from  $P([n], m)$  and conditioned on the index  $i$  being included in the set of indices  $J$ . As a result, the first term that we need to estimate is equal to

$$\alpha_{i,1} := \Pr_{I \sim P([n] \setminus \{i\}, m-1), h \leftarrow \mathcal{A}(S_{I \cup \{i\}})}[h(x) = y] = \Pr_{J \sim P([n], m), h \leftarrow \mathcal{A}(S_J)}[h(x) = y \mid i \in J].$$

This implies that instead of sampling from  $P([n] \setminus \{i\}, m-1)$  for every  $i$  separately, we can use samples from  $P([n], m)$ , select the samples for which  $J$  contains  $i$  and evaluate this term on them.

Specifically, given  $J_1, \dots, J_{t/2}$  sampled randomly from  $P([n], m)$  we use  $\mathcal{A}$  to train models  $h_1, \dots, h_{t/2}$  on each of the dataset  $S_{J_1}, \dots, S_{J_{t/2}}$ . Now for every  $i$ , we can estimate  $\alpha_{i,1}$  as

$$\mu_{i,1} = \frac{|\{k \in [t/2] : i \in J_k, h_k(x) = y\}|}{|\{k \in [t/2] : i \in J_k\}|},$$

or set  $\mu_{i,1} = 1/2$  if the denominator is equal to 0.

Similarly, the distribution of  $I$  sampled from  $P([n] \setminus \{i\}, m-1)$  is the same as the distribution of  $J$  sampled from  $P([n], m-1)$  and conditioned on the index  $i$  being excluded from  $J$ . Therefore the second term that we need to estimate is equal to:

$$\alpha_{i,2} := \Pr_{I \sim P([n] \setminus \{i\}, m-1), h \leftarrow \mathcal{A}(S_I)}[h(x) = y] = \Pr_{J \sim P([n], m-1), h \leftarrow \mathcal{A}(S_J)}[h(x) = y \mid i \notin J].$$

This means that we can estimate the second term analogously by sampling  $J_{t/2+1}, \dots, J_t$  from  $P([n], m-1)$ , using  $\mathcal{A}$  to train models  $h_{t/2+1}, \dots, h_t$  on each of the resulting subsets and then estimating the second term as

$$\mu_{i,2} = \frac{|\{t/2+1 \leq k \leq t : i \notin J_k, h_k(x) = y\}|}{|\{t/2+1 \leq k \leq t : i \notin J_k\}|},$$

or set  $\mu_{i,2} = 1/2$  if the denominator is equal to 0. The final estimator is defined for every  $i \in [n]$  as  $\mu_i = \mu_{i,1} - \mu_{i,2}$ .

We now compute the expected squared error of each of the terms of this estimator. For  $\mu_{i,1}$  we consider two cases. The case in which the denominator  $|\{k \in [t/2] : i \in J_k\}|$  is equal to at least  $\frac{mt}{4n}$  and the case in which the denominator is less than  $\frac{mt}{4n}$ . In the first case we are effectively estimating the mean of a Bernoulli random variable using the empirical mean of least  $\frac{mt}{4n}$  independent samples. The expectation of each of the random variables is exactly equal to  $\alpha_{i,1}$  and thus the squared error is exactly the variance of the empirical mean. For a Bernoulli random variable this means that it is equal to at most  $\frac{4n\alpha_{i,1}(1-\alpha_{i,1})}{mt} \leq \frac{n}{mt}$ . For the second case, note that for every  $k, i \in J_k$  with probability  $m/n$ . Therefore the multiplicative form of the Chernoff bound for the sum of  $t/2$  independent Bernoulli random variables implies that the probability of this case is at most  $e^{-\frac{mt}{16n}}$ . Also note that in this case we are either estimating the mean using  $\ell < \frac{mt}{4n}$  independent samples or using the fixed value 1/2. In both cases the squared error is at most 1/4. Thus

$$\mathbf{E}[(\alpha_{i,1} - \mu_{i,1})^2] \leq \frac{e^{-\frac{mt}{16n}}}{4} + \frac{n}{mt}.$$

An analogous argument for the second term gives

$$\mathbf{E}[(\alpha_{i,2} - \mu_{i,2})^2] \leq \frac{e^{-\frac{(n-m+1)t}{16n}}}{4} + \frac{n}{(m-n+1)t}.$$

By combining these estimates we obtain that

$$\begin{aligned} \mathbf{E}[(\text{inf1}_m(\mathcal{A}, S, i, z) - \mu_i)^2] &\leq \mathbf{E}[(\alpha_{i,1} - \mu_{i,1})^2] + \mathbf{E}[(\alpha_{i,2} - \mu_{i,2})^2] \\ &\leq \frac{n}{mt} + \frac{n}{(m-n+1)t} + \frac{e^{-\frac{mt}{16n}}}{4} + \frac{e^{-\frac{(n-m+1)t}{16n}}}{4} \\ &\leq \frac{1}{pt} + \frac{1}{(1-p)t} + \frac{e^{-pt/16}}{2}, \end{aligned}$$

where we used that  $p = \min(m/n, 1 - m/n)$ . □

**Remark A.1.** In practice, models trained on random subsets of size  $m - 1$  are essentially identical to models trained on random subsets of size  $m$ . Thus, in our implementation we improve the efficiency by a factor of 2 by only training models on subsets of size  $m$ . Our estimator also benefits from the fact that for most examples, the variance of each sample  $\alpha_{i,1}(1 - \alpha_{i,1})$  (or  $\alpha_{2,1}(1 - \alpha_{2,1})$ ) is much smaller than  $1/4$ . Finally, it is easy to see that the estimator is also strongly concentrated around  $\text{inf1}_m(\mathcal{A}, S, i, z)$  and the concentration result follows immediately from the concentration of sums of independent Bernoulli random variables.

## B Details of the Experimental Setup

We implement our algorithms with Tensorflow [Aba+15]. We use single NVidia® Tesla P100 GPU to for most of the training jobs, except for ImageNet, where we use 8 P100 GPUs with single-node multi-GPU data parallelization.

We use ResNet50 [He+16] in both ImageNet and CIFAR-100 experiments, which is a Residual Network architecture widely used in the computer vision community [He+16]. Because CIFAR-100 images ( $32 \times 32$ ) are smaller than ImageNet images ( $224 \times 224$ ), for CIFAR-100 we replace the first two layers (a convolution layer with  $7 \times 7$  kernel and  $2 \times 2$  stride, and a max pooling layer with  $3 \times 3$  kernel and  $2 \times 2$  stride) with a single convolution layer with  $3 \times 3$  kernel and  $1 \times 1$  stride. We use data augmentation with random padded (4 pixels for CIFAR-100 and 32 pixels for ImageNet) cropping and random left-right flipping during training. For MNIST, we use a simplified Inception [Sze+15] model as described in [Zha+17].

We use stochastic gradient descent (SGD) with momentum 0.9 to train the models. For ImageNet, we use batch size 896 and base learning rate 0.7. During the 100 training epochs, the learning rate is scheduled to grow linearly from 0 to the maximum value (the base learning rate) during the first 15 epochs, then it remains piecewise constant, with a  $10 \times$  decay at epoch 30, 60 and 90, respectively. Our implementation achieves  $\approx 73\%$  top-1 accuracy when trained on the full training set.

We also use SGD with momentum 0.9 for CIFAR-100 training. To achieve faster training, we use slightly larger batch size (512) and base learning rate (0.4) than usual. During the 160 training epochs, the learning rate is scheduled to grow linearly from 0 to the maximum value (base learning rate) in the first 15% iterations, and then decay linearly back to 0 in the remaining iterations. Our implementation achieves  $\approx 76\%$  top-1 accuracy when trained on the full training set. In the experiment on the estimation consistency, we also trained CIFAR-100 on a number of different architectures. ResNet18 and Inception are trained using exactly the same hyper-parameter configuration as described above. For DenseNet, we halved the batch size and learning rate due to higher memory load of the architecture. The linear models on pre-computed hidden representations are also trained using the same hyper-parameter as ResNet50, except they train for only 40 epochs due to fast convergence.

For MNIST, we use SGD with momentum 0.9 and the same learning rate scheduler as the CIFAR-100 experiment, with base learning rate 0.1. We train the models for 30 epochs with batch size 256.

Our ImageNet training jobs takes about half an hour for each training epoch. On CIFAR-100, the training time per epoch is about: 1 minute and 30 seconds for ResNet50, 17 seconds for ResNet18, 45 seconds for DenseNet100, 14 seconds for Inception, and 0.5 second for Linear models on pre-computed hidden representations. Our training time on MNIST is about 7 seconds per epoch.

Our architectures and training algorithms are not state-of-the-art since state-of-the-art training is significantly more computationally intensive and it would not be feasible for us to train thousands of models.

## C Selection Procedure for Examples of Influence Estimates

For our influence figures, to avoid cherry-picking, we select the training examples to include as follows. We first sort the training examples in the selected high-influence pairs by the highest influence they have on a test example. We then pick 3 consecutive sets each of 5 training examples with indices spread evenly in the sorted order. The exact Python code is included below.

```
n_copies = 3
n_egs = 5
```

```
idx_sort_selected = np.argsort(-max_infl_of_train_selected)

base_idxs = np.linspace(0, len(idx_train_selected) - n_copies, n_egs).astype(np.int)
for i_copy in range(n_copies):
    idxs_to_depict = [idx_train_selected[idx_sort_selected[x + i_copy]] for x in base_idxs]
    visualize_tr_examples_and_influence(idxs_to_depict, n_test_egs=4)
```