UNDER REVIEW

RESCIENCE C

Editorial

[Re] Learning with Noisy Labels Revisited: A Study Using Real-World Human Annotations

Žiga Rot[1, ID], Valter Hudovernik[1, ID], Klemen Vovk[1, ID], Luka Škodnik[1, ID], and Luka Čehovin Zajc[1, ID]

[1]University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, 1000 Ljubljana

Edited by
(Editor)

Received
01 November 2024

Published
–

DOI
–

1 Introduction

Learning with noisy labels (LNL) is a machine-learning problem where some labels in the training data are misassigned. The sources of misassignment may differ, but generally, LNL is most often associated with human labeling errors. LNL poses a significant challenge to conventional learning algorithms, which assume that the provided labels are reliable. When this assumption is violated, models may learn erroneous patterns, resulting in poor performance and biased predictions. Approaches that explicitly address the issue of noisy labels became popular with the rise of deep learning and focus on modifications of the iterative learning process, as well as the observation that correct patterns are learned faster than those that accommodate noisy samples [1].
The LNL is primarily evaluated on computer vision datasets dealing with object classification tasks. Each image has a class label that has to be predicted. In the LNL scenario, a part of the labels in the training split of the dataset is considered to be corrupted. The training problem, shared among all LNL approaches, consists of two key components: an underlying classifier model and a learning strategy on top of it. As with any good experimental setup, it is important to keep as many parts of the training procedure fixed as possible when comparing the performance of different learning strategies. This way, we can confirm that the differences stem entirely from the proposed changes and are not the consequence of other modifications (e.g., using a model with higher capacity).

2 Scope of Reproducibility

Our project aims to reproduce the results of the first attempt to compare several approaches using a dataset with realistically generated noisy labels, described in [2]. The authors propose a relabelled CIFAR dataset [3] that includes human annotation errors, named CIFAR-N. They examine the differences between artificially obtained synthetic and human-error-based noise and benchmark 20 different LNL approaches.
To promote the use of their newly annotated datasets, the authors claim that the real-world noises in their datasets significantly differ from synthetically generated noises. Specifically, they claim that training on real-world noise is a harder problem than training on their respective synthetic counterparts. To prove this, they conduct the following series of experiments, which we try to reproduce:

1. They first conduct a qualitative and quantitative evaluation of differences between the human noises and synthetic noises generated from the same transition matrices. They first cluster the samples based on input variables and then construct

Correspondence should be addressed to ()
The authors have declared that no competing interests exist.
Code is available at https://github.com/rescience-c/template. – SWH .

transition vectors in each cluster. After qualitatively evaluating that the transition vectors differ between real-world and synthetic noises, they confirm the observation via a null hypothesis test.

2. They train a ResNet-34 [4] classifier on real-world noises and synthetic noises generated from the same transition matrices and show that the model starts memorizing the real-world labels faster than the synthetic ones, indicating a harder learning task.

3. They claim to reproduce 20 of, at the time, state-of-the-art LNL methods, run them on real-world noises and synthetic noises generated from the same transition matrices, and evaluate their performance. They show that the methods achieve lower test accuracies on a clean hold-out set when trained with real-world noise, again showing that learning on real-world noise is harder.

We reproduced the first two points and reached the same conclusions, albeit with slightly different results. To try to verify the third point, we reimplement the 20 methods into a shared framework. Since the authors do not include the details of the hyperparameter setup for each LNL method, we must also reverse-engineer all the hyperparameter setups used in the benchmark. This takes significant time due to long run times, so we focus only on a subset of 10 methods for the benchmark experiment. We cannot reproduce the reported results using the methodology described by the authors. However, we reach similar results using the original hyper-parameters for each LNL strategy. Our source code is publicly available at https://github.com/KlemenVovk/noisy-labels.

3 Methodology

We follow the authors' methodology for all experiments except when reproducing their LNL benchmark, which we justify in Section 3.4. Where the procedures are not well described in the original publication or are ambiguous, we describe our selected methodology in greater detail.

3.1 Datasets

We use the CIFAR-N datasets provided by the benchmark authors [2]. Their work is based on the standard CIFAR-10 and CIFAR-100 datasets [3], consisting of $50,000$ train and $10,000$ test images with image-level class annotations. As the names suggest, CIFAR-10 includes 10 label categories, while CIFAR-100 has 100.
The authors propose noisy extensions for both datasets, which keep the original images but replace the labels with real-world noisy label sets. Only the training sets of both datasets have these, while the $10,000$ testing labels are kept from the original CIFAR-10 and -100 datasets and are considered noise-free. Both datasets also include the $50,000$ training labels from the original CIFAR datasets as ground truth clean labels.
The CIFAR-10N extension includes five sets of noisy labels obtained from different annotators with varying degrees of noise. Specifically, the authors present the following label sets:

- **Random1, Random2, Random3**: These directly include the labels submitted by different annotators. These have a medium noise level, with 17.23%, 18.12%, and 17.64% noise ratios, respectively.

- **Aggregate**: Labels in this set are obtained from Random1, Random2 and Random3 via majority voting. Whenever there is a 3-way disagreement, the label is selected randomly. Due to the aggregation, the noise level here is the lowest at 9.03%.

- **Worse**: Similarly, as in the Aggregate set, labels here are also obtained from the three Random sets but are aggregated instead, so incorrect labels are taken wherever possible. As such, the noise level here is the highest, with 40.21% incorrect labels.

Similarly, the CIFAR-100N extension includes two sets of real-world noisy labels. Besides the original 100 categories, the authors also present a new set of coarse annotations of 20 super-classes to help the annotators when assigning one of the 100 fine classes. The annotators are first asked to assign a super-class, from which they then later select the fine-grained class. The two noisy label sets are then defined as follows:

- **Coarse**: This label set consists of the 20 super-class labels obtained from the annotators. It has a noise rate of 25.60%.
- **Fine**: This one consists of all 100 fine categories and has 40.20% of noisy labels.

3.2 Noise Hypothesis Testing

The authors of [2] qualitatively and quantitatively show that real-world human noise differs from synthetic noise with hypothesis testing. We follow the same procedure but take note of certain ambiguous decisions.
We first start by fitting a ResNet34 model on clean CIFAR-10 train labels. The authors do not explicitly explain how the model was trained or what input augmentations were used. We use the default training methodology for the CE baseline described by the authors, which we present in Section 3.5. After the training, we use the classifier to get image embeddings for all $50,000$ train images. We do so by extracting them from a module before the final linear layer, as described by the authors.
The experiment is based on the Random1 label set, for which we generate its synthetic counterpart. We do so by constructing a transition matrix $T, T_{i,j} = \mathbb{P}(\widetilde{Y} = j|Y = i)$, where $\widetilde{Y}$ is the noisy set and Y is the clean label set. We use T to generate a set of synthetic labels from the clean label set.
Next, for each of the $K = 10$ classes, we cluster its embeddings into $\nu = 5$ clusters using a K-Means model. As all cluster members belong to the same clean class, we can then obtain a K-dimensional transition vector for each cluster: $\mathrm{p}_{k,v} = [\mathbb{P}(\widetilde{Y} = j|Y = k, X \in C_v)]_{j\in[K]}$. We do this for both real-world and synthetic labels. At this point, we also plot the resulting transition vectors in each class and cluster and qualitatively check for differences between the real-world and synthetic transition vectors.
We repeat the previous steps $E = 10$ times, generating a new set of synthetic labels and embeddings each time. This gives us a set of transition vectors $\{\mathrm{p}_{e,k,v}\}_{e\in[E],k\in[K],v\in[\nu]}$ for both synthetic and human noise. From this data, we can obtain the two sets of l_2 distances between the transition vectors – the human-synthetic distances:

$$\{d^{(1)}_{e,k,v} = ||\mathrm{p}^{human}_{e,k,v} - \mathrm{p}^{synthetic}_{e,k,v}||^2_2\}_{e\in[E],k\in[K],v\in[\nu]}$$

and the synthetic-synthetic distances:

$$\{d^{(2)}_{e,k,v} = ||\mathrm{p}^{synthetic}_{e,k,v} - \mathrm{p}^{synthetic}_{(e+1) \mod E,k,v}||^2_2\}_{e\in[E],k\in[K],v\in[\nu]}.$$

Following the procedure in [2], we perform a two-sided T-test to validate the hypothesis.

3.3 Noise Memorisation

To further investigate the differences between artificial and real-world noise, the authors of [2] inspect the learning behavior of a classifier trained on both sets of noisy labels. Specifically, they compare the learning behavior on Aggregate, Random1, and

Worse sets of real-world labels with their synthetic counterparts, generated from their respective transition matrices, as described in Section 3.2.
The experiment setup is not explained in [2] beyond the fact that the ResNet-34 [4] is used as a classifier. From the figures in the original paper, we have deduced that each experiment was run for 150 epochs. As mentioned in Section 5.4, we lost contact with the authors before they could describe the details of this experiment. We assume the other hyperparameters are consistent with the benchmark training methodology presented in Section 3.5.2.
We then log the number of memorized labels at each epoch to inspect the memorization behavior. The $50,000$ training samples are split into two disjoint sets based on whether the noisy label matches the clean one or not. Then, for both sets, we note the ratio of samples that have been memorized in each epoch, where a sample x is considered memorized by the classifier f if the following holds: $\exists i \in [K]$ s.t. $\mathbb{P}(f(x) = i) > 0.95$, where K is the number of classes.

3.4 Benchmarking Learning Strategies

Like the original authors, we reimplement 20 methods for learning with noisy labels in a unified framework. Based on the original works and implementations, we reimplement and integrate the following methods: *Forward/Backward-T* [5], *GCE* [6], *Co-teaching (+)* [7, 8], *T-revision* [9], *Peer loss* [10], *ELR (+)* [11], *Divide-Mix* [12], *JoCoR* [13], *Cores*^{2} [14], *VolMinNet* [15], *CAL* [16], *PES (semi)* [17], *SOP (+)* [18], *Positive-LS* [19] and *F-divergence* [20] where (+) denotes that both the *method* and *method*+ versions are included. Due to the large computation cost, we select 10 methods for which we are most certain that our implementations are correct: *CE* (baseline), *Co-Teaching* [7], *Co-Teaching*+ [8], *ELR, ELR*+ [11], *Divide-Mix* [12], *VolMinNet* [15], *CAL* [16], *PES (semi)* [17], *SOP* and *SOP*+ [18]. Here, we note that the author's reported performance for the SOP method is most likely SOP+, so we include both methods in our benchmark.
The benchmark authors do not describe the evaluation procedure for the benchmark at all. Therefore, we must piece it together from GitHub issues and our correspondence. It proceeds as follows. Each learning strategy is trained for 100 epochs, wherein, in each epoch, the underlying classifier is tested on the clean test set in terms of micro-averaged classification accuracy. After the training is completed, the highest test accuracy among all epochs is reported. All this is repeated five times for each method, from which the final mean and standard deviations for the accuracy scores are then reported. Due to long computation times, we reduce the number of repetitions to three in our experiments.
We note that fixing the learning rate schedule, optimizer, and the number of training epochs is not the best way to compare different LNL strategies, as these are integral aspects of some methods. Instead, a more sensible approach to make the results comparable would be to use the same classifier (e.g., ResNet34) for all the methods, which the authors do not do. For this reason, we first run our experiments on a single noisy label set using the methodology described by the authors. The discrepancy between the reported and our obtained results is shown in Table 2. For this reason, we use the original configurations of each LNL method. With these, we are able to reproduce the results reported by the benchmark authors.

3.5 Experimental Setup and Code

Comparing different learning strategies only makes sense if the underlying classifier is fixed. If not, the differences between any two methods could stem only from the larger classifier capacity. Even worse, the inferior strategy could outperform a superior one only because the underlying model is better. To address this issue, the authors claim to have fixed the classifier to a ResNet34 where possible.
As such, we at first use the reference ResNet34 [4] implementation from PyTorch. Setting up the baseline cross-entropy training, we realize the results differ from the per-

formance reported in the benchmark. Moreover, substituting the PyTorch implementation of ResNet34 with the benchmark authors' one yields significantly higher accuracy. Looking further into the matter, despite seeming like the model's code was exactly the same as in PyTorch's codebase, we notice that the authors' implementation changes the first convolution layer and skips the first max-pooling operation, making feature maps in each layer 16 times larger than in the original. As the final linear layers have access to more features, this difference significantly boosts the performance, which could, at first glance, be entirely attributed to the learning strategy. Therefore, we find fixing the same underlying classifier important when comparing the methods.
We observe the same pattern of nearly identical ResNet implementations in most of the learning strategies' codebases. However, some strategies use different backbones, which presents a question of whether reported performance can be attributed to the proposed LNL strategy or the changes in the underlying architecture. The reimplemented framework is modular enough to combine different models and LNL strategies and enables more consistent and controllable evaluation. Since the main goal of the paper is to reproduce the results of [2], different LNL strategies are still configured to use either original ResNet-34 or, where applicable, pre-activation ResNet-18 variations (both with the altered first convolutional layer and without the first max-pooling layer).
We implement our framework using PyTorch[1] and Lightning[2] libraries, as they enable a high level of abstraction, thus reducing potential sources of error while also being flexible enough to support even the more complex among the tested learning strategies and enabling robust settings for reproducibility and easy distributed training. The framework is modular, enabling potential future LNL experiments.

Verifying our LNL Implementations — Our decision to re-implement the entire experiment, including the evaluated LNL methods, requires additional verifications. We adopt an alternative verification strategy, which includes the following steps. Firstly, an original, or the most reliable, unofficial implementation of the method is found. We use implementations linked in the original papers, or the popular implementations listed on the Papers With Code website. We then run the reference implementation several times using default parameters on existing datasets. After we integrate the method into our framework, we rerun it several times using the same parameter configuration and dataset as in the original codebase. This includes all the hyperparameters, optimizers, learning rate schedulers, backbone, etc. For successful verification, the original loss and accuracy curves must match the ones of the original implementation. During our verification of the original methods, we also observe some possible inconsistencies, which we describe in Section 5.1.

Hyper-parameters — While being a crucial part of many learning strategies, the benchmark authors do not discuss the hyper-parameters for each entry in full detail. Therefore, we decide to use the configurations presented in the original papers of each LNL method as a base.
What the benchmark authors claim to have fixed, we also fix. This includes the total training time of 100 epochs, a stochastic gradient descent optimizer with an initial learning rate of 0.1, and a step learning rate scheduler with a single step at the 60th epoch, where the learning rate is multiplied by a factor $\gamma = 0.1$. We use the benchmark authors' version ResNet-34 for most of the learning strategies, and a pre-activation ResNet-18 where applicable. Since we cannot match the performance in the benchmark with this setup, we rerun all the experiments with the original number of epochs, optimizers, and learning rate schedulers provided by the individual learning strategy authors. We list the full hyper-parameter setup in Appendix B.

[1] https://pytorch.org/
[2] https://lightning.ai/docs/pytorch/stable/

3.6 Computational Requirements

We run all our experiments on a system with 8 nVidia Titan X Pascal GPUs running Ubuntu 20.04.2 LTS and Cuda version 12.2. With this setup, it takes approximately 1.5 GPU hours to reproduce the noise hypothesis testing experiment and approximately 27 GPU hours to reproduce the noise memorization experiment. Reproducing benchmark results on the selected subset of the LNL methods takes 52.5 GPU hours per label set; we present a breakdown of the training times in Table 1. It would take approximately 2,000 GPU hours to reproduce our results fully (eight label sets, each with three repeats, and six synthetic runs with two repeats).

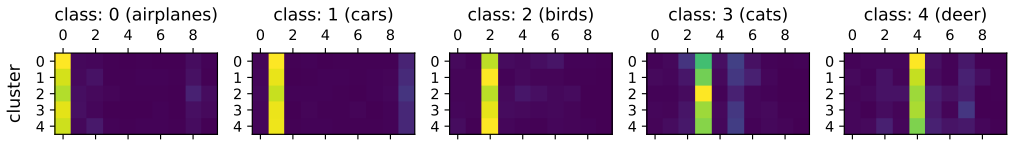
Method	Runtime (h)
CE	1.5
Co-teaching (+)	5.6
ELR	1.8
ELR+	7.9
DivideMix	14.0
VolMinNet	1.3
CAL	1.5
PES (semi)	12.2
SOP+	5.5
SOP	1.2
total	52.5

Table 1. Runtimes for different methods, for a single repeat of a single noise label.

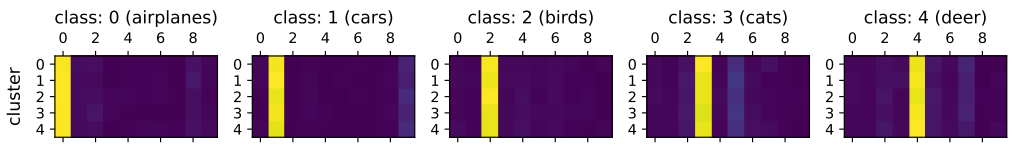
4 Results

In this section, we describe the results obtained using the methodology previously described. Each section corresponds to one of the reproducibility claims outlined in Section 2.

4.1 Noise Hypothesis Testing



(a) Human noise (Random 1): noise level $\approx 17.23\%$.



(b) Synthetic class-dependent noise with the same T.

Figure 1. Transition vectors within $\nu = 5$ clusters for the first 5 classes for real-world human noise and corresponding synthetic noise. Qualitatively, human noise shows a greater diversity of transition vectors between the clusters in each class.

We confirm the original claim that human noisy labels differ from synthetic ones, even when generated by the same transition matrix. We do not match the obtained p-value entirely, with ours being higher at $3.5 \cdot 10^{-18}$ versus $1.8 \cdot 10^{-36}$, presented in [2]. We attribute the difference to possibly different training regimes. Qualitatively, the difference between the human and synthetic cluster transition vectors is also clearly visible in Figure 1, where transition vectors in synthetic clusters are more uniform than in the real-world scenario, which stems from the fact that the synthetic noise is not feature dependent.

4.2 Noise Memorisation

Figure 2 shows the noise memorization history for three different label sets. The models start memorizing the real-world noisy labels faster than the synthetic ones. The effect is not as apparent as presented in [2], but the same conclusion can be reached. Again, the difference can be attributed to a different training configuration, as the authors did not provide all the details in the paper.

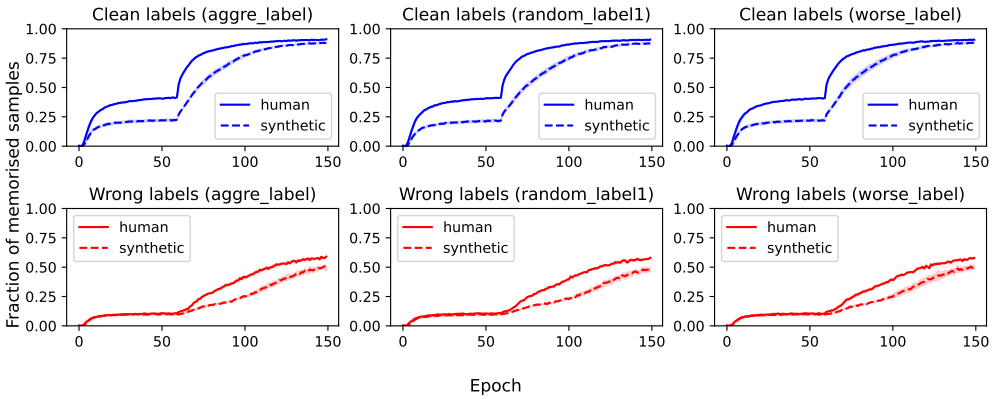


Figure 2. Noise memorization effects. The models start to memorize the human noisy labels (full line) faster than the synthetic ones (dashed line).

4.3 Benchmark

Method	Described Config	Original Config	Reported
CE	91.70 ± 0.07	91.70 ± 0.07	87.77 ± 0.38
Co-teaching	90.25 ± 0.13	91.93 ± 0.25	91.20 ± 0.13
Co-teaching+	86.20 ± 0.88	91.15 ± 0.08	90.61 ± 0.22
ELR*	93.00 ± 0.19	93.00 ± 0.19	92.38 ± 0.64
ELR+*	95.32 ± 0.06	95.32 ± 0.06	94.83 ± 0.10
DivideMix*	95.62 ± 0.09	95.62 ± 0.09	95.01 ± 0.71
VolMinNet	84.02 ± 6.61	90.47 ± 0.17	89.70 ± 0.21
CAL	91.76 ± 0.22	91.84 ± 0.23	91.97 ± 0.32
PES (semi)	91.53 ± 0.44	94.64 ± 0.05	94.66 ± 0.18
SOP+	93.17 ± 0.66	96.04 ± 0.15	95.61 ± 0.13

Table 2. Comparison of our results using the authors' methodology, our results using the original configurations for each LNL, and the authors' reported results on the CIFAR-10-N Aggregate label set. Only the results for which the use of original configurations was reported match our implementation (ELR (+) and DivideMix) denoted by *.

Table 2 shows the results obtained using the evaluation methodology described in [2]. One can notice significant differences between the results reported in the original paper and those obtained when following the methodology fully (fixing the same optimizer, learning rate schedule, and number of epochs for all methods). Upon further investigation, the experiments were rerun using the original (different) configuration for each LNL method. The results were closer to the ones reported in [2] this time.
We reproduce the authors' results on the CIFAR-N dataset using the configurations specific to each evaluated method. We compare our results with results reported in [2] (Tables 2 and 8). Table 3 shows the results of the reimplemented methods for all label sets in CIFAR-N.

Method	CIFAR-10N						CIFAR-100N	
	Clean	Aggregate	Random 1	Random 2	Random 3	Worst	Clean	Noisy
CE	94.21 ± 0.12	91.70 ± 0.07	90.20 ± 0.04	90.12 ± 0.13	90.08 ± 0.05	83.91 ± 0.08	76.23 ± 0.19	61.19 ± 0.51
Co-teaching	92.15 ± 0.11	91.93 ± 0.25	90.69 ± 0.07	90.51 ± 0.14	90.56 ± 0.22	80.92 ± 0.43	72.24 ± 0.44	54.48 ± 0.27
Co-teaching+	92.90 ± 0.22	91.15 ± 0.08	89.82 ± 0.13	89.64 ± 0.19	89.77 ± 0.26	82.36 ± 0.04	70.39 ± 0.45	55.46 ± 0.34
ELR	93.97 ± 0.12	93.00 ± 0.19	92.20 ± 0.10	92.05 ± 0.12	92.18 ± 0.17	87.89 ± 0.14	75.64 ± 0.21	63.72 ± 0.38
ELR+	95.81 ± 0.16	95.32 ± 0.06	94.89 ± 0.11	94.88 ± 0.08	94.93 ± 0.07	91.75 ± 0.06	$\mathbf{78.82 \pm 0.24}$	67.87 ± 0.07
DivideMix	95.51 ± 0.00	95.62 ± 0.09	$\mathbf{95.72 \pm 0.11}$	$\mathbf{95.78 \pm 0.10}$	95.71 ± 0.09	$\mathbf{93.10 \pm 0.10}$	78.22 ± 0.06	$\mathbf{70.91 \pm 0.09}$
VolMinNet	92.71 ± 0.02	90.47 ± 0.17	88.90 ± 0.51	88.81 ± 0.17	88.67 ± 0.10	80.87 ± 0.25	72.73 ± 0.65	58.30 ± 0.05
CAL	93.78 ± 0.18	91.84 ± 0.23	91.10 ± 0.26	90.60 ± 0.10	90.61 ± 0.12	84.82 ± 0.23	74.53 ± 0.21	60.13 ± 0.33
PES (semi)	94.75 ± 0.16	94.64 ± 0.05	95.20 ± 0.08	95.26 ± 0.13	95.20 ± 0.11	92.58 ± 0.05	77.77 ± 0.33	70.32 ± 0.28
SOP+	$\mathbf{96.53 \pm 0.05}$	$\mathbf{96.04 \pm 0.15}$	95.70 ± 0.07	95.62 ± 0.18	$\mathbf{95.75 \pm 0.14}$	92.89 ± 0.09	77.90 ± 0.29	63.88 ± 0.32
SOP	93.76 ± 0.26	91.69 ± 0.76	89.83 ± 0.24	90.57 ± 0.46	90.88 ± 0.15	83.66 ± 0.64	72.97 ± 1.15	56.17 ± 1.02

Table 3. Results based on the original works' configurations. Results for most of the methods match those reported in the original work.

For most of the methods, we are able to reproduce the results reported by the authors. One notable difference is that the baseline CE method performs much better in our experiments. The discrepancy between our and original baseline results is investigated in Appendix A. Some methods use noise rate as an input parameter [7, 8, 12]. The parameter is fixed to the value obtained from the original implementations. The justification for such a decision is that one does not always know how many labels are noisy in a real-world scenario. This way, the experimental protocol remains consistent, and the comparison is fair to all the methods. The decision to fix the noise level parameter could explain the difference in the Co-teaching(+) methods' performance on the CIFAR-100N noisy label set. Similarly, in the case of SOP(+) methods, symmetric noise type was assumed as an input parameter, which might explain the gap in performance on CIFAR-100N.

Human Noise vs. Synthetic Noise — Following the protocol described in [2], an experiment using synthetic noise produced by the transition matrices of CIFAR-N datasets is performed. The test accuracy on the synthetic labels is then subtracted from those trained on the human labels. The results are shown in Table 4.
We can see that LNL methods largely perform better on synthetic data. This is at least the case for CIFAR-10N, while for CIFAR-100N, the results are split evenly, with 5 methods performing better and 5 methods performing worse. This matches observations reported in [2], indicating that learning on real-world noise is more difficult to handle.

5 Discussion

We managed to reproduce the hypothesis testing and noise memorization effects. Due to a partially unclear description of the experiment protocol, we do not obtain the same results as the authors, but the results nonetheless lead to the same conclusions. The real-world label noise is indeed different from its synthetic counterpart, and the classifiers start to overfit on it faster, which indicates a harder learning task.
While the authors describe the benchmarking experiment in greater detail, we fail to reach the same results using their methodology. Instead, when we use the original

Method	CIFAR-10N					CIFAR-100N
	Aggregate	Random 1	Random 2	Random 3	Worst	Noisy
CE	0.71 ± 0.10	0.93 ± 0.39	1.01 ± 0.26	1.10 ± 0.05	3.06 ± 0.21	1.89 ± 0.51
Co-teaching	0.89 ± 0.27	0.46 ± 0.08	0.09 ± 0.15	0.34 ± 0.23	1.40 ± 0.85	-2.26 ± 0.36
Co-teaching+	1.05 ± 0.16	1.24 ± 0.14	1.12 ± 0.19	1.19 ± 0.34	2.92 ± 0.31	-1.94 ± 1.10
ELR	0.12 ± 0.21	0.31 ± 0.18	0.50 ± 0.13	0.31 ± 0.18	-3.82 ± 0.91	-0.71 ± 0.49
ELR+	0.20 ± 0.08	0.31 ± 0.13	0.32 ± 0.08	0.33 ± 0.08	-0.66 ± 2.48	-0.28 ± 0.10
DivideMix*	0.56 ± 0.10	0.47 ± 0.12	0.54 ± 0.10	0.44 ± 0.11	1.94 ± 0.11	1.08 ± 0.48
VolMinNet	0.89 ± 0.17	1.15 ± 0.51	1.14 ± 0.18	1.41 ± 0.11	3.68 ± 0.67	3.11 ± 0.15
CAL	-0.09 ± 0.37	-0.76 ± 0.31	-0.23 ± 0.36	-0.25 ± 0.19	-0.34 ± 0.23	-0.57 ± 0.33
PES (semi)	0.29 ± 0.30	0.31 ± 0.10	0.41 ± 0.23	0.39 ± 0.11	1.71 ± 0.26	0.09 ± 0.77
SOP+	0.25 ± 0.16	0.38 ± 0.08	0.48 ± 0.18	0.45 ± 0.15	2.07 ± 0.09	0.89 ± 0.36
SOP	0.70 ± 0.99	1.20 ± 0.83	0.17 ± 0.46	0.86 ± 0.29	2.73 ± 1.29	2.44 ± 1.02

Table 4. Differences between the real-world and synthetic test accuracies: $\text{acc}_{\text{syn}} - \text{acc}_{\text{real}}$. Negative gaps representing the method performed better when training on human noise are highlighted in red. For CIFAR-10N, most of the methods perform better on synthetic noise, indicating a harder learning task. For CIFAR-100N, this is not the case.

hyperparameter configurations for each LNL method, we land closer to the reported accuracies for most of the methods. Perhaps most interesting is that the baseline cross-entropy training performs significantly better than expected, outperforming several methods designed explicitly for LNL scenarios even in the most difficult noise settings. This behavior also persists outside our framework, even in the original code provided by the authors, as we show in Appendix A.

5.1 Additional Observations

During our efforts to reproduce the paper and the benchmarked LNL methods, we have observed details in the methodology that we do not agree with or peculiarities in the implementations that could potentially benefit researchers working on LNL problems.
Firstly, when benchmarking LNL methods, the authors use a clean test set to measure accuracy every epoch, selecting the highest one at the end. In a real-world LNL setup, this would not be feasible since it is assumed that the data is noisy in an unknown way, with no notion of which labels are clean and which are not. This methodology then overestimates the real-world performance, obstructing a fair comparison between the LNL methods.
Secondly, one must be careful when using Table 3 when comparing the performance of different LNL strategies. As discussed in Section 3.5, the underlying classifier greatly contributes to the model's accuracy. One should not directly compare the performances of LNL strategies with differing model architectures, since the difference might greatly depend on the choice and changes to the architecture. This does not change the conclusions of [2] for the importance of the difference between real-world and synthetic noise, which was confirmed in our experiment reproduction. However, it becomes an issue when we use the same results as a benchmark of LNL strategies. A more decoupled view between the LNL strategy and the architecture choice would be desired in this case.
Lastly, we notice during our implementation of several LNL methods that the original implementations incorrectly generate synthetic noise. This brings their reported performances into question. The symmetric noise produced by the official DivideMix [12] implementation [3] uses a biased noise rate. For a given noise rate r, their implementation randomly selects a fraction r of clean labels. The noisy labels are sampled uniformly from all classes k. This obviously results in $\frac{1}{k}$ of the samples agreeing with the original label, resulting in the actual noise rate being $\widetilde{r} = r - \frac{1}{k}r$ instead of r. This is especially

[3] https://github.com/LiJunnan1992/DivideMix/blob/39740b267147466bac0779de91d969909ab3b65f/dataloader_cifar.py#L58-L74

evident in high-noise scenarios where DivideMix claims to work best. We observe an even bigger error in the ELR [11] implementation of synthetic noise. The method uses two models trained on separate datasets, which are both noised independently, while also producing noisy labels with a decreased noise rate $\widetilde{r}$ described above [4]. Here, without loss of generality, we treat the sample as noisy and as having the wrong label in the first dataset. This results in the probability of at least one of the models observing the correct label for a sample given that the sample is noisy, being $1 - \widetilde{r}$ instead of 0.
This brings into question the results reported in the respective original works; however, for the purposes of our benchmark reproduction where human noise is used, the results are unaffected, and both of the methods rank among the top-performing ones.

5.2 What was easy

Using the starter code and the datasets provided by the benchmark authors was easy and only required minimal tweaking to run. Running most of the LNL strategies' repositories was also easy enough, as most authors include instructions for basic use-case and experiment reproduction in the source code repository.

5.3 What was difficult

Throughout the reproduction attempt, we encountered several problems, most of them stemming from unclear and ambiguous descriptions of the benchmark in [2]. Many evaluated methods rely on algorithm-specific hyperparameters, which the authors did not list. Therefore, an effort was made to try to recover them by experimentally trying out different combinations in an attempt to match the reported results. The learning rate schedule for all methods was said to follow a multi-step schedule with a single decay at the 50th epoch. In the original source code, the switch happens at the 60th epoch.
Some methods require access to an additional noisy validation set to perform model selection or infer noise model parameters. The authors do not describe the use of a validation set in the original paper or our correspondence. Because of this, we discarded many of the methods from the benchmark as there were too many unknown factors that needed to be inferred for experiments to be fair between the benchmark entries – e.g., do we use the validation set in all methods or only in those that require it, and use all the data to train the rest.

5.4 Communication with Original Authors

We contacted the authors two times during our implementation effort. The first time we contacted them, we inquired about several inconsistencies between the paper and the provided code. We also asked about the hyperparameter selection protocol, which backbone models were used, which checkpoints were selected, and some general evaluation and method-specific questions. The authors responded with a short email, answering some of our questions but avoiding the first significant deviations from their results.
After some time, we contacted them for the second time. We inquired how the validation sets were handled, as some methods could not be properly trained without them. We also asked about the specifics of the hypothesis testing setup and the disagreement with the results. This time, the authors did not respond.

[4] https://github.com/shengliu66/ELR/blob/909687a4621b742cb5b8b44872d5bc6fce38bdd3/ELR/data_loader/cifar10.py#L73-L80

References

1. D. Arpit, S. Jastrzębski, 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: **International conference on machine learning**. PMLR. 2017, pp. 233–242.
2. J. Wei, Z. Zhu, H. Cheng, T. Liu, G. Niu, and Y. Liu. "Learning with Noisy Labels Revisited: A Study Using Real-World Human Annotations." In: **International Conference on Learning Representations**. 2022.
3. A. Krizhevsky, G. Hinton, et al. "Learning multiple layers of features from tiny images." In: (2009).
4. K. He, X. Zhang, S. Ren, and J. Sun. **Deep Residual Learning for Image Recognition**. 2015. arXiv:1512.03385 [cs.CV].
5. G. Patrini, A. Rozza, A. Menon, R. Nock, and L. Qu. **Making Deep Neural Networks Robust to Label Noise: a Loss Correction Approach**. 2017. arXiv:1609.03683 [stat.ML].
6. Z. Zhang and M. R. Sabuncu. "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels." In: **CoRR** abs/1805.07836 (2018). arXiv:1805.07836.
7. B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. W. Tsang, and M. Sugiyama. "Co-sampling: Training Robust Networks for Extremely Noisy Supervision." In: **CoRR** abs/1804.06872 (2018). arXiv:1804.06872.
8. X. Yu, B. Han, J. Yao, G. Niu, I. W. Tsang, and M. Sugiyama. "How does Disagreement Help Generalization against Label Corruption?" In: **CoRR** abs/1901.04215 (2019). arXiv:1901.04215.
9. X. Xia, T. Liu, N. Wang, B. Han, C. Gong, G. Niu, and M. Sugiyama. **Are Anchor Points Really Indispensable in Label-Noise Learning?** 2019. arXiv:1906.00189 [cs.LG].
10. Y. Liu and H. Guo. **Peer Loss Functions: Learning from Noisy Labels without Knowing Noise Rates**. 2020. arXiv:1910.03231 [cs.LG].
11. S. Liu, J. Niles-Weed, N. Razavian, and C. Fernandez-Granda. "Early-Learning Regularization Prevents Memorization of Noisy Labels." In: **CoRR** abs/2007.00151 (2020). eprint: 2007.00151.
12. J. Li, R. Socher, and S. C. H. Hoi. "DivideMix: Learning with Noisy Labels as Semi-supervised Learning." In: **CoRR** abs/2002.07394 (2020). arXiv:2002.07394.
13. H. Wei, L. Feng, X. Chen, and B. An. **Combating noisy labels by agreement: A joint training method with co-regularization**. 2020. arXiv:2003.02752 [cs.CV].
14. H. Cheng, Z. Zhu, X. Li, Y. Gong, X. Sun, and Y. Liu. **Learning with Instance-Dependent Label Noise: A Sample Sieve Approach**. 2021. arXiv:2010.02347 [cs.LG].
15. X. Li, T. Liu, B. Han, G. Niu, and M. Sugiyama. "Provably End-to-end Label-Noise Learning without Anchor Points." In: **CoRR** abs/2102.02400 (2021). arXiv:2102.02400.
16. Z. Zhu, Y. Song, and Y. Liu. "Clusterability as an Alternative to Anchor Points When Learning with Noisy Labels." In: **CoRR** abs/2102.05291 (2021). arXiv:2102.05291.
17. Y. Bai, E. Yang, B. Han, Y. Yang, J. Li, Y. Mao, G. Niu, and T. Liu. "Understanding and Improving Early Stopping for Learning with Noisy Labels." In: **CoRR** abs/2106.15853 (2021). arXiv:2106.15853.
18. S. Liu, Z. Zhu, Q. Qu, and C. You. "Robust Training under Label Noise by Over-parameterization." In: **Proceedings of the 39th International Conference on Machine Learning**. Ed. by K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 14153–14172.
19. M. Lukasik, S. Bhojanapalli, A. K. Menon, and S. Kumar. **Does label smoothing mitigate label noise?** 2020. arXiv:2003.02819 [cs.LG]. URL: https://arxiv.org/abs/2003.02819.
20. J. Wei and Y. Liu. **When Optimizing f-divergence is Robust with Label Noise**. 2021. arXiv:2011.03687 [cs.LG]. URL: https://arxiv.org/abs/2011.03687.

A Baseline Performance Results

The biggest discrepancy between our and the authors' results comes from the baseline (CE) training. In [2], the exact procedure for obtaining their results is not described. After investigating the official implementation and contacting the authors, a decision was made to keep the checkpoint with the best test accuracy. However, this procedure resulted in the discrepancy between our reproduced results and the ones reported in the original work [2]. The discrepancy is visualized in Figure 3. We can see that the accuracy of the best checkpoint in our experiments is higher than their reported performance across all runs. We were not alone in noticing this discrepancy, at the time of investigation, there were two open issues on the authors' official Github repository with the same question. Authors of the issues report the last obtained accuracies on the

Worst label set:$\approx$ 68% [5] and $\approx$ 67% [6], which are in line with our last epoch accuracy (68 ± 0.88).

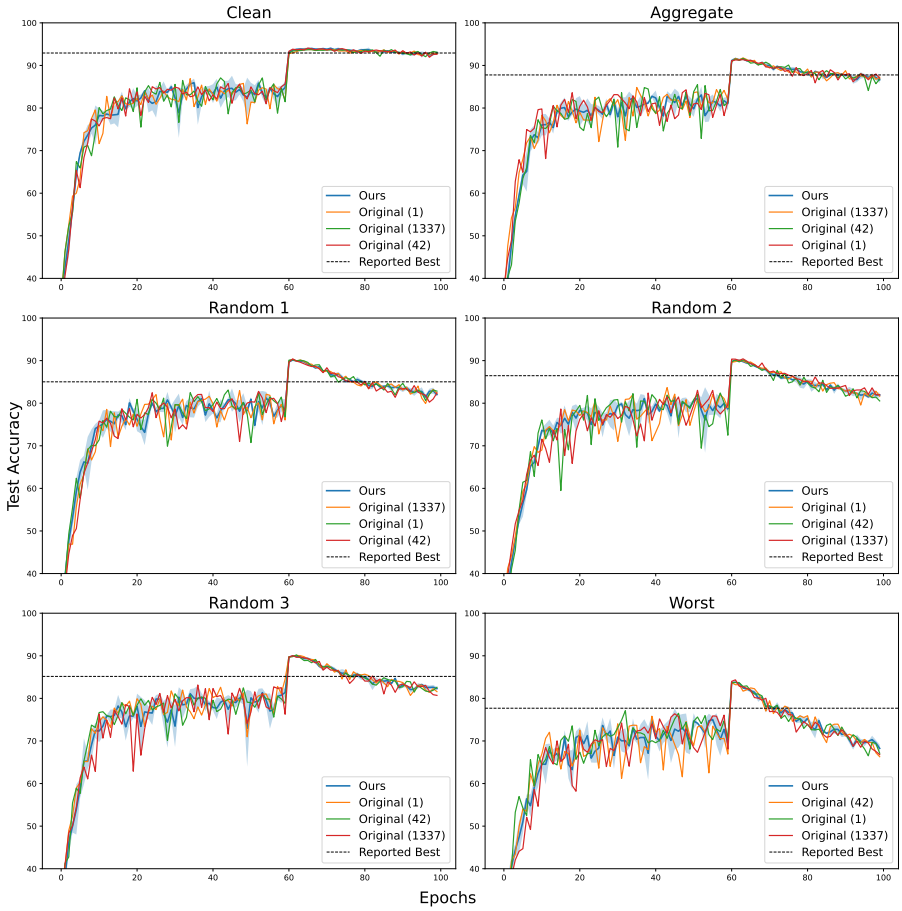


Figure 3. Differences between the authors' reported (dotted) baseline performance and accuracy obtained in reproduction (full blue) as well as using the original code (orange, red, yellow). The performance obtained when using a reimplemented framework perfectly aligns with the original code. However, on all the noise labels, their reported best accuracy does not coincide with any of the runs.

B Hyperparameters

In this section, we report the hyperparameters for all the methods included in our reproduced evaluation. Table 5 describes the hyperparameters of methods using the *PreActResNet18* backbone, and Table 6 for the methods using the *ResNet34* backbone. Here we again note that most of the methods' original implementations as well as the model implementation provided by the authors [7] use ResNet implementations that use stride of 1 instead of 2 in the first layer, resulting in a four times increase in activation volumes. Their modified implementation also excludes the first max pooling layer, resulting in another four times increase in activation volumes, for a total of 16 times bigger activation volumes. In all of our experiments, we use this version of ResNet provided in the author's original repository.

[5] https://github.com/UCSC-REAL/cifar-10-100n/issues/5
[6] https://github.com/UCSC-REAL/cifar-10-100n/issues/8
[7] https://github.com/UCSC-REAL/cifar-10-100n/blob/main/models/resnet.py

Method	Hyperparameter	CIFAR10N	CIFAR100N
DivideMix	optimizer	SGD	SGD
	lr	0.02	0.02
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	LambdaLR	LambdaLR
	epochs	300	300
	alpha	4	4
	noise_type	asymmetric	asymmetric
	p_thresh	0.5	0.5
	temperature	0.5	0.5
	lambda_u	0	0
	warmup_epochs	10	30
ELR+	optimizer	SGD	SGD
	lr	0.02	0.02
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	MultiStepLR ([150], 0.1)	MultiStepLR ([200], 0.1)
	epochs	200	250
	beta	0.7	0.9
	lmbd	3	7
	alpha	1	1
	gamma	0.997	0.997
	ema_step	40000	40000
	coef_step	0	40000
SOP+	optimizer	SGD	SGD
	lr	0.02	0.02
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	CosineAnnealing	CosineAnnealing
	epochs	300	300
	lr_u	10	1
	lr_v	10	10
	overparam_mean	0.0	0.0
	overparam_std	1e-08	1e-08
	ratio_balance	0.1	0.1
	ratio_consistency	0.9	0.9

Table 5. Hyperparameters for methods using PreActResNet18 backbone.

Method	Hyperparameter	CIFAR-10N	CIFAR-100N
CAL	optimizer	SGD	SGD
	lr	0.1	0.1
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	MultiStepLR ([60], 0.1)	MultiStepLR ([60], 0.1)
	epochs	165	165
	alpha	0.0	0.0
	alpha_scheduler	seg	seg
	warmup_epochs	65	65
	alpha_list_warmup	[0.0, 2.0]	[0.0, 1.0]
	milestones_warmup	[10, 40]	[10, 40]
	alpha_list	[0.0, 1.0, 1.0]	[0.0, 1.0, 1.0]
	milestones	[10, 40, 80]	[10, 40, 80]
CE	optimizer	SGD	SGD
	lr	0.1	0.1
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	MultiStepLR ([60], 0.1)	MultiStepLR ([60], 0.1)
	epochs	100	100
Co-teaching	optimizer	Adam	Adam
	lr	0.001	0.001
	weight_decay	0	0
	scheduler	alpha_schedule	alpha_schedule
	epochs	200	200
	forget_rate	0.2	0.2
	exponent	1	1
	num_gradual	10	10
	epoch_decay_start	80	100
Co-teaching+	optimizer	Adam	Adam
	lr	0.001	0.001
	weight_decay	0	0
	scheduler	alpha_schedule	alpha_schedule
	epochs	200	200
	init_epoch	20	5
	forget_rate	0.2	0.2
	exponent	1	1
	num_gradual	10	10
	epoch_decay_start	80	80
ELR	optimizer	SGD	SGD
	lr	0.02	0.02
	weight_decay	0.001	0.001
	momentum	0.9	0.9
	scheduler	CosineAnnealing	MultiStepLR ([80, 120], 0.01)
	epochs	120	150
	beta	0.7	0.9
	lmbd	3	7
PES (semi)	optimizer	SGD	SGD
	lr	0.02	0.02
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	CosineAnnealing	CosineAnnealing
	epochs	300	300
	PES_lr	0.0001	0.0001
	warmup_epochs	20	35
	T2	5	5
	lambda_u	5	75
	temperature	0.5	0.5
	alpha	4	4
SOP	optimizer	SGD	SGD
	lr	0.02	0.02
	weight_decay	0.0005	0.0005
	momentum	0.9	0.9
	scheduler	MultiStepLR ([40, 80], 0.1)	MultiStepLR ([40, 80], 0.1)
	epochs	120	150
	lr_u	10	1
	lr_v	10	10
	overparam_mean	0.0	0.0
	overparam_std	1e-08	1e-08
	ratio_balance	0.0	0.0
	ratio_consistency	0.0	0.0
VolMinNet	optimizer	SGD	SGD
	lr	0.01	0.01
	weight_decay	0.0001	0.0001
	momentum	0.9	0.9
	scheduler	MultiStepLR ([30, 60], 0.1)	MultiStepLR ([30, 60], 0.1)
	epochs	80	80
	lam	0.0001	0.0001
	init_t	2	2
	optimizer_transition_mtx	SGD	Adam

Table 6. Hyperparameters for methods using ResNet34 backbone.