A Model-Agnostic Approach for Learning with Noisy Labels of Arbitrary Distributions

Shuang Hao[1,2*] Peng Li[3] Renzhi Wu[3] Xu Chu[3]
[1]School of Computer and Information Technology, Beijing Jiaotong University, China
[2]Beijing Key Laboratory of Traffic Data Analysis and Mining, China
[3]Georgia Institute of Technology, USA
haoshuang@bjtu.edu.cn, {pengli@, renzhiwu@, xu.chu@cc.}gatech.edu

***Abstract*—Most real-world datasets contain label noise, which can negatively affect downstream ML models trained on them. To deal with this problem, one can clean the mislabeled data before training, which is not only time-consuming and expensive but also requires domain expertise. Another approach is to use a noise-robust ML training algorithm. However, existing methods have some prerequisites that may not be practical in many applications (*e.g.,* they are tied to specific downstream model architecture or they are applicable to specific noise distributions).**

In this paper, we propose a model-agnostic approach for learning with noisy labels of arbitrary distributions. In particular, our approach can work with any gradient descent optimization based machine learning model and deal with any label noise distribution. We achieve them by proposing two theoretically grounded noise-robust loss functions (for different noise distributions), and we are able to automatically decide which loss function to use based on a novel noise setting detection module. We directly learn the required hyper-parameters in the loss functions via meta-learning technique to minimize the loss on a given small clean validation set, and propose several strategies to improve the efficiency of training. Experiments on multiple datasets with both real-world and injected label noise show that our method performs better than state-of-the-art approaches.

I. Introduction

Data quality is of critical importance especially in the new era of data-driven artificial intelligence (AI), since the nature of the input data can strongly influence the results coming from these AI systems. However, practically collected training data often contain noisy labels [1], [2]. Many studies have shown that the presence of label noise would hurt model performance and increase model complexity [3], [4], [5], [6]. Deep neural networks (DNNs) can be particularly susceptible to noisy labels because of their high capability to memorize the labels including incorrect ones [7]. Zhang et al. [8] proved that DNNs can easily fit an entire training dataset with any ratio of noisy labels, which eventually results in the model's poor generalizability.

Frénay et al. [9] characterized noise generation in terms of its distribution and proposed three possible statistical models of label noise. Figure 1 shows two of them which are more likely to occur in reality. Here, the dots of different colors represent the data of different labels and Figure 1(a) shows the state of no noisy labels. The label noise in Figure 1(b) is class-dependent, where the occurrence of label error fully depends on the true class labels. Thus we can observe that the purple dots as noisy data are evenly distributed in the orange region. And in this situation, a noise transition matrix is enough to demonstrate the probabilities of different label errors. In contrast, the label noise in Figure 1(c) is feature-dependent, where the occurrence of label error depends on both features and true labels. We can observe that the purple dots as noisy data are located near the boundary (Figure 1(c-1)) or concentrated in a specific internal area (Figure 1(c-2)) of the orange region and each sample has its own noise transition vector.

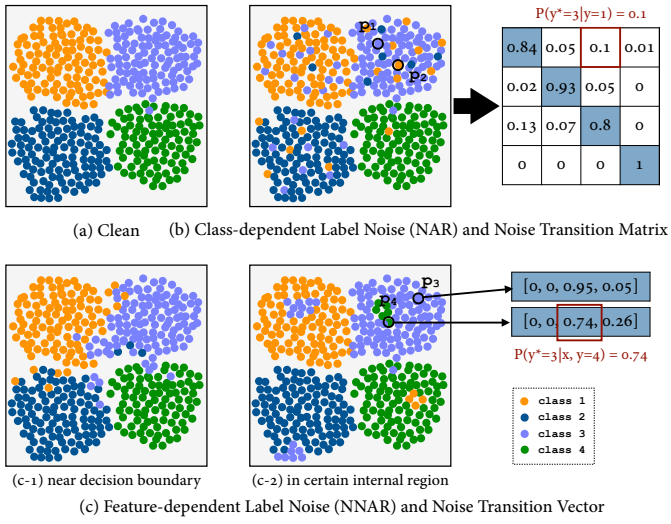


Fig. 1. Label Noise Distribution

Existing Approaches and Limitations. Various methods have been proposed to deal with label noise. The first category of solutions is to identify all unreliable labels, then relabel or remove them before training starts [10], [11]. However, relabelling can be costly, and some deletion methods may remove a substantial amount of data, including lots of difficult but important examples, which may downgrade the model performance. The second category of approaches is to build a specific noise-tolerant model [12], [13], [14]. However, this model-specific approach is not easily applicable if users would like to use other ML models.

The third category of approaches, which are model-agnostic, is to use a noise-robust loss function [2], [15], [16]. One of

* Work partly done while visiting Georgia Tech.

the representative methods is the loss correction [15]. This approach assumes that the label noise is class-dependent. Then a noise transition matrix T can be used to correct the model prediction or the loss of each training example. T is either assumed to be known or estimated by polishing a subset of the training data. However, in the real-world scenarios, label noise would usually be more complicated and tend to be feature-dependent.

Our Proposal. In this paper, we propose a model-agnostic approach for learning with noisy labels of arbitrary distributions. Specifically, we propose to use two noise-robust loss functions J_c and J_f, and prove that our loss functions are unbiased (*i.e.,* the minimization of J_c or J_f on noisy data is equivalent to the minimization of the traditional loss J on clean data). J_c is for the class-dependent noise setting where the probability of label error only depends on the true class and the label noise distribution can be modeled by a noise transition matrix T. J_f is mainly for the feature-dependent noise setting where the probability of label error also depends on the example's features (and hence can model arbitrary label noise distributions). In this situation, we create the transition vector for each training example to perform its own loss correction.

Technical Challenges. There are three practical challenges to be addressed when utilizing J_c and J_f to build the noise-robust model. The first challenge is *how to learn the noise transition matrix and transition vectors accurately as the hyper-parameters to correct the loss*. Some previous work have studied how to estimate the transition matrix T. For example, Patrini et al. [15] assumed that there exists "perfect example" of each class which has the largest probability of belonging to that class, and the estimation of each row of T can just be based on the prediction of each "perfect example"; Northcutt et al. [17] counted examples that are likely to belong to another class to generate T. Different from the above work, we directly learn the noise transition matrix/vector from the data during model training via meta-learning paradigm, which is a nested optimization process where an outer loop optimizes meta-parameters controlling the optimization of model parameters within an inner loop. Concretely, we assume that the best transition matrix/vector should minimize the loss of an unbiased clean validation set that are consistent with the evaluation procedure. Towards this goal, we regard the transition matrix/vector as meta-parameters to learn in the outer loop to minimize the loss on the clean validation set, while the model training in the inner loop is to minimize the loss on the noisy training set.

The second challenge is *how to decide which one to use, J_c or J_f, given a real-world dataset with unknown label noise distribution*. Many existing approaches assume that the label noise is under a certain known distribution [1], [15], [18]. However, the label noise distribution in real-world datasets is usually unknown. This is also an issue in our work: we need to choose a proper loss function for a given dataset based on its noise distribution. *Our key observation is that the noisy examples will be more evenly distributed in the data space when the probability of label error only depends on the true class*. This observation is mainly from the definition of label noise settings. The class-dependent label noise only depends on the true label and is independent from features. Thus, the noisy examples are evenly distributed in the feature space. In contrast, if the distribution of noisy examples also depends on features, noisy data will be concentrated in a specific region of feature space. Exploiting this observation, we design a statistical test to determine the noise setting. Our null hypothesis is the simpler setting: label noise only depends on the true class which means noisy data points are evenly distributed. Under this hypothesis, the number of noisy neighbors of each data point is expected to be the same and the variance of the number of noisy neighbors for all data points should be small. When this hypothesis is not true, noisy data points are not evenly distributed and tend to form noisy clusters, thus the number of noisy neighbors of each data point can differ significantly and the variance the number of noisy neighbors for all data points will be large. By comparing the observed variance to the distribution of the variance obtained under null hypothesis, we can decide whether the null hypothesis (*i.e.,* class-dependence assumption) is true.

The third challenge is *how to efficiently learn the hyper-parameters in J_c and J_f and avoid overfitting*. If we have N training examples divided into C classes, there will be $C \times C$ hyper-parameters in J_c to learn from the clean validation set and this number will increase substantially to $N \times C$ in J_f. Moreover, it is worth noting that in reality we cannot obtain a lot of clean examples as the validation data. Therefore, it is necessary to study how to improve the training efficiency with lots of parameters and hyper-parameters to learn and avoid overfitting to the validation set. In this paper, we propose several strategies to achieve this goal including the fast calculation of gradients. Particularly, for J_f, we observe that if we pre-train a model with dirty data using a higher learning rate and early-stopping regularization, the training examples that can be correctly predicted are almost with correct labels since they are the "easier" ones being first remembered. Therefore, there is no need to do loss correction for them, which can reduce the hyper-parameters to learn by at least a half.

Summary of Contributions. In summary, as shown in Figure 2, given a set of training examples with noisy labels but unknown noise distribution and a small but clean validation set, we first detect the label noise setting on hand. If it is the class-dependent noise setting, we train a model Φ_c which minimizes the training loss J_c as well as the traditional validation loss J (e.g., the cross entropy loss) via meta-learning paradigm. Otherwise, we will train a model Φ_f that minimizes the training loss J_f and the validation loss J to make the model robust to the feature-dependent label noise. Our paper makes the following contributions:

- We propose two noise-robust loss functions J_c and J_f, and prove that they are unbiased (*i.e.,* the minimization of J_c or J_f on noisy data is equivalent to the minimization of the

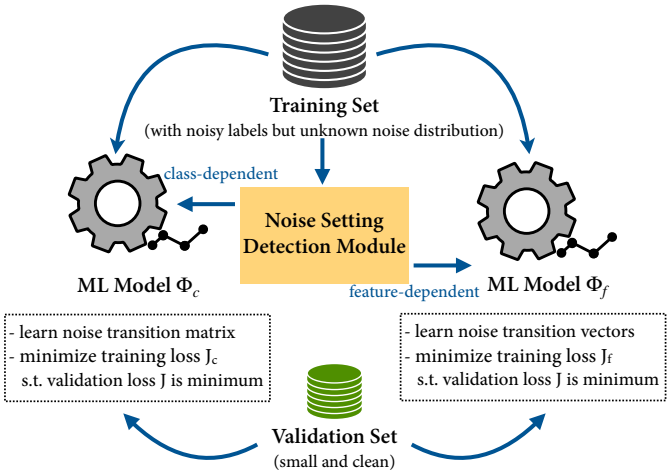


Fig. 2. Framework Overview

traditional loss on clean data).

- We leverage an additional small and clean validation set to adaptively learn the noise transition matrix/vectors via meta-learning paradigm.
- We present an effective method to decide the label noise setting of training data and some strategies to improve the training efficiency, all of which reduce the time to obtain the desired model.
- We conducted extensive experiments on both image and text datasets with real-world and injected label noise. The results show that our method outperforms other state-of-the-art approaches.

Organization. The rest of this paper is organized as follows. Section II introduces the related work. Section III formally defines the label noise and our studied problem. We discuss the class-dependent noise setting in Section IV and feature-dependent noise setting in Section V, including their corresponding loss function, learning procedure and efficiency problem. Section VI presents an effective method to decide the label noise setting, complementing our framework. Experimental results are shown in Section VII followed by the conclusion (Section VIII).

II. Related Work

Learning Directly with Noisy Labels. Some methods can remain relatively effective even if the training data is polluted by little label noise, *e.g.,* random forest and other ensemble classifier [19], [20], [21]. Overfitting avoidance techniques such as regularization can also be used to partially handle label noise [22], [23]. However, it has been proven that common loss functions are not robust to label noise [24], while our two novel loss functions can have a good performance even when the noise rate is high.

Label Noise Reduction. Many data cleaning methods are proposed to deal with the label-polluted datasets, such as removing or relabeling the incorrect labels before training occurs. Outlier and anomaly detection techniques are widely used to discover the mislabeled examples [25], [26], and other no-human-involved label cleaning algorithms rely on the predictions of classifiers learning from the noisy data [17], [27]. The suspicious examples can also be resubmitted to an expert for relabeling [28], further can be used as the seed examples to detect and remove more mislabeled data [29]. In our paper, we do not remove suspicious training examples but automatically correct the loss of these data examples without losing any useful information.

Weighting Examples. Weighting can be used to emphasize clean examples and weaken noisy ones, where mislabelled examples are expected to be assigned with smaller weights [30], [31], [32], [33]. The weighting strategies in previous work [30], [31] aim to minimize the weighted training loss and validation loss respectively. Wang et al. [32] assigned a weight for each example based on the confidence supplied by noisy label detection module where feature similarities were compared, and Jiang et al. [33] employed another neural network called MentorNet to provide the sample weighting scheme and supervised the training of the base deep networks. Instead of heuristically re-weighting examples under cross entropy loss, we correct the loss of dirty data using two novel loss functions with a theoretical guarantee.

Robust Loss Function. Traditional machine learning models can become noise-tolerant with robust loss function. Some work [34], [16] present the conditions that the robust loss function should meet and examine some of the popular loss functions for learning neural networks. Zhang et al. [2] combined the cross entropy loss and mean absolute loss to be noise-robust. Loss correction approaches are also utilized in noise-robust learning [1], [15], [35], which mostly use example predictions to estimate the noise transition matrix that can lead to noise-robust performance. Our work also falls into this category. We utilize a meta-learning paradigm to optimize the loss correction and learning model parameters together rather than separately handling them. Different from other meta-learning framework [36] that adopts an extra network to learn how to correct labels under cross entropy loss, our method directly corrects the loss via two novel loss functions with theoretical guarantee and does not need extra networks.

Noise Modeling with Deep Learning. A label noise model is built to describe the relationships among features, noisy labels and true labels, and further integrated into a deep learning system to detect and correct the wrong labels. Some previous work [37], [38] introduce an extra linear layer at the end of the neutral network to learn the noise distribution. Xiao et al. [12] extended CNNs with a probabilistic graphical model, inferring the true labels to supervise the training of the network. Class embedding vectors are learnt in existing work [39] with an attention mechanism and transfer learning to predict the relevance of an example to its noisy class label. Instead, our method will not change the model structure or utilize an extra noise-aware mechanism to deal with label errors, which make it easier to be deployed.

III. Problem Formulation

Label Noise. In supervised C-class classification, one has a feature space $\mathcal{X} \subseteq \mathbb{R}^d$ and a label space $\mathcal{Y} = \{1, ..., C\}$. Each data point $(\mathbf{x}, y)$ with feature $\mathbf{x} \in \mathcal{X}$ is tagged to one label $y \in \mathcal{Y}$ to indicate which class it belongs to. *label noise* refers to that observed labels are incorrect, *i.e.,* the observed label y is different from the true label $y^* \in \mathcal{Y}$.

Taxonomy of Label Noise. There are three possible statistical models of label noise [9]:

(1) *Noisy completely at random model (NCAR)*: the occurrence of an incorrect label is independent of the feature $\mathbf{x}$ and true class y^*. In NCAR, the observed label is different from the true class with a certain error rate $p(y \neq y^*)$.

(2) *Noisy at random model (NAR)*: the probability of label error depends on the true class y^*. This is also referred as the *class-dependent noise setting*. In NAR, label noise can be modeled with conditional probability $p(y = j|y^* = i)\ \forall i, j$, namely the probability that the observed label is j when the true label should be i. The conditional probabilities of all possible i and j form a corruption transition matrix T^* where $\mathsf{T}^*_{ij} = p(y = j|y^* = i)$ that fully defines the label corruption process.

(3) *Noisy not at random model (NNAR)*: label noise depends on both the feature $\mathbf{x}$ and the true class y^*. This is also referred as the *feature-dependent noise setting*. In this setting, mislabeling is more likely for certain classes and certain regions of feature space. This can be formally expressed as $p(y = i|\mathbf{x}, y^*)$, i.e. the probability of observing a particular label i depends on both the feature $\mathbf{x}$ and the true class y^*.

It is impractical to assume that label errors occur completely randomly. In the real world, examples from certain classes may be more likely to be mislabeled (e.g. mules are likely to be mislabeled as horse); examples located near the decision boundary can also be easily mislabeled. Thus, we only consider the NAR and NNAR settings in this paper.

Problem Statement. Given a set of training data $D_\eta = \{(\mathbf{x}_i, y_i), i \in [1, N]\}$ with label noise, an ML model $\Phi(\mathbf{x}, \theta)$ with parameter θ, a small clean validation set $D_v = \{(\mathbf{x}_j, y_j), j \in [1, M]\}$ and a test set D_t, the goal is to maximize the accuracy on the test set of model Φ after is it trained on D_η.

Remark. Following other meta-learning-based frameworks [35], [36], [40], we assume that there exists a small set of validation data with clean labels. Typically, the clean validation set we need is much smaller than the noisy training set ($M \ll N$), thus we can have an expert manually check each label in the validation set.

IV. NAR Noise Setting

We first study the simpler case where the probability of label error only depends on the class, in which case the label noise can be modeled simply by a transition matrix. We introduce our robust loss function for the NAR noise setting in Section IV-A. Then, we present our method of learning with NAR noise in Section IV-B and give more algorithmic details in Section IV-C.

A. Loss Function for NAR Noise

Given a training example $(\mathbf{x}, y) \in D_\eta$, let $l(\mathbf{x}, y; \theta)$ or $l(y, h_\theta(\mathbf{x}))$ denote its training loss. If the cross entropy loss is adopted, $l(y, h_\theta(\mathbf{x})) = \hat{y} log(h_\theta(\mathbf{x}))$ where $\hat{y}$ is the one-hot representation of class y and $h_\theta(\mathbf{x})$ is the predicted probabilities from the model. When the observed label is not certainly clean, we need to consider all the possible values of its true label, so the loss of an example $(\mathbf{x}, y)$ should be $l_c(\mathbf{x}, y; \theta) = \sum_{y^* \in \mathcal{Y}} p(y^*|y) \times l(\mathbf{x}, y^*; \theta)$. Similarly, we can define a recovering transition matrix T where $\mathsf{T}_{yy^*} = p(y^*|y)$ describes how to recover the correct label from the noisy label (the connection between T^* and T is that $\mathsf{T}_{yy^*} p(y) = \mathsf{T}^*_{y^*y} p(y^*)$). With the recovering matrix T, the loss function can be rewritten as:

$$J_c = \frac{1}{N} \sum_{(\mathbf{x},y) \in D_\eta} \sum_{y^* \in \mathcal{Y}} \mathsf{T}_{yy^*} \times l(\mathbf{x}, y^*; \theta). \tag{1}$$

J_c is a linear combination of the loss for each possible true label where the weights of the combination are the probabilities of the truth labels given the observed label y. We can show that the minimization of J_c on noisy data is equivalent to the minimization of the vanilla loss J on clean data.

Theorem 1: J_c is an unbiased loss function for the NAR noise setting, i.e.,

$$\forall \mathbf{x},\ \mathbb{E}_{\mathbf{x},y} l_c(\mathbf{x}, y; \theta, \mathsf{T}) = \mathbb{E}_{\mathbf{x},y^*} l(\mathbf{x}, y^*; \theta).$$

Therefore, we have

$$\text{argmin}_\theta\ \mathbb{E}_{\mathbf{x},y} l_c(\mathbf{x}, y; \theta, \mathsf{T}) = \text{argmin}_\theta\ \mathbb{E}_{\mathbf{x},y^*} l(\mathbf{x}, y^*; \theta).$$

Proof of Theorem 1:

$$\begin{aligned}
\mathbb{E}_{\mathbf{x},y} l_c(\mathbf{x}, y; \theta, \mathsf{T}) &= \mathbb{E}_{\mathbf{x},y} \sum_{y^* \in \mathcal{Y}} \mathsf{T}_{yy^*} \times l(\mathbf{x}, y^*; \theta) \\
&= \int \sum_{y \in \mathcal{Y}} p(\mathbf{x}, y) \sum_{y^* \in \mathcal{Y}} p(y^*|y) l(\mathbf{x}, y^*; \theta) d\mathbf{x} \\
&= \int \sum_{y^* \in \mathcal{Y}} \sum_{y \in \mathcal{Y}} p(\mathbf{x}, y) p(y^*|y) l(\mathbf{x}, y^*; \theta) d\mathbf{x} \\
&= \int \sum_{y^* \in \mathcal{Y}} p(\mathbf{x}, y^*) l(\mathbf{x}, y^*; \theta) d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x},y^*} l(\mathbf{x}, y^*; \theta)
\end{aligned}$$

Our loss J_c has a similar form with the loss correction approach [15]. In that work they model the corruption transition matrix T^*, while we directly model the recovering transition matrix T. In their case, they have to assume T^* to be non-singular and the inverse of T^* has to be calculated which is numerically unstable. Our loss J_c makes no such assumption and does not incur any numerical problem.

B. Learning with NAR Noise

The effectiveness of the above loss function highly depends on the estimate of the noise transition matrix T. In this paper, we regard T as hyper-parameter. Naively tuning the hyper-parameter T requires training the model many times which

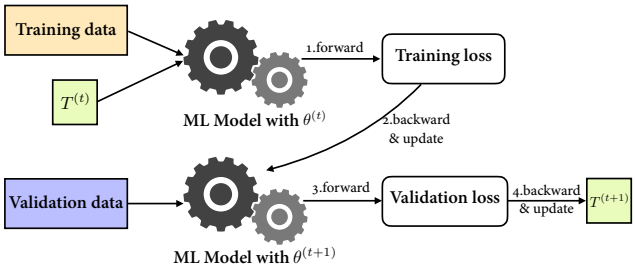


Fig. 3. Framework of Model Training

is extremely computationally expensive. We instead propose a method to optimize T and model parameter θ at the same time so we only need to train the model for one time.

Optimization of θ. At every step t of training, a mini-batch $\{X_\eta, Y_\eta\} = \{(\mathbf{x}_i, y_i), i \in [1, n]\}$ of the training set is visited, where n is the size of the training batch and $n \ll N$. Here, we fix T and optimize the model parameters according to the descent direction of the training loss J_c on the mini-batch

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} \left(\frac{1}{n} \sum_{i=1}^{n} l_c(\mathbf{x}_i, y_i; \theta^{(t)}, \mathsf{T}^{(t)})\right), \tag{2}$$

where α is the learning rate of the model.

Optimization of T. After updating model parameters (fixed $\theta^{(t+1)}$), we sample a mini-batch $\{X_v, Y_v\} = \{(\mathbf{x}_j^v, y_j^v),\ j \in [1, m]\}$ from the validation set where m is the batch size, and optimize T based on the validation loss on the mini-batch

$$\tilde{\mathsf{T}}^{(t+1)} = \mathsf{T}^{(t)} - \gamma \nabla_{\mathsf{T}^{(t)}} \left(\frac{1}{m} \sum_{j=1}^{m} l(\mathbf{x}_j^v, y_j^v; \theta^{(t+1)})\right), \tag{3}$$

where γ is the learning rate of optimizing T.

Note that T is a matrix of probabilities, so each value should be in $[0, 1]$ and $\sum_j \mathsf{T}_{ij} = \sum_j p(y^* = j | y = i) = 1$. We have to normalize $\tilde{\mathsf{T}}^{(t+1)}$: we first clamp all elements in $\tilde{\mathsf{T}}^{(t+1)}$ to be non-negative values, *i.e.,* $\tilde{\mathsf{T}}_{ij}^{(t+1)} = max(\tilde{\mathsf{T}}_{ij}^{(t+1)}, 0)$, then normalize it to values in $[0, 1]$ by dividing by row-wise sum:

$$\mathsf{T}_i^{(t+1)} = \frac{\tilde{\mathsf{T}}_i^{(t+1)}}{\sum \tilde{\mathsf{T}}_i^{(t+1)} + \delta(\tilde{\mathsf{T}}_i^{(t+1)})}, \tag{4}$$

where $\tilde{\mathsf{T}}_i^{(t+1)}$ and $\mathsf{T}_i^{(t+1)}$ denote the ith row of $\tilde{\mathsf{T}}^{(t+1)}$ and $\mathsf{T}^{(t+1)}$ respectively, and $\delta(\cdot)$ is to prevent division by zero.

Framework Overview. The framework of learning with J_c is shown in Algorithm 1 and depicted by Figure 3. The inputs of Algorithm 1 include the dirty training set D_η, the clean validation set D_v, the initialized model parameter θ. The output is the model that can achieve the best validation accuracy in the training process. The hyperparameters include the batch size of training data n and validation data m; the number of epochs E to convergence; the model learning rate α and transition matrix learning rate γ, which can be decided via cross validation.

We first initialize the noise transition matrix T to an identity matrix where the elements in the diagonal are ones and the rest are zeros, indicating that no label noise is considered at first (line 1). In each iteration, we sample one batch of data $\{X_\eta, Y_\eta\}$ from the training set, fix T to compute the training loss l_η and update the model parameters θ (line 5-9). Then, we randomly sample one batch $\{X_v, Y_v\}$ from the validation set and test the model performance on it (line 11-12). The corresponding validation loss l_v is used to update T for the next iteration (line 13-15). We return the parameter θ_{best} which can achieve the best validation accuracy (line 16-17).

Algorithm 1: Framework of Learning with J_c

Input: training set D_η with N examples, validation set D_v, training batch size n, validation batch size m, model parameter θ, the number of epochs E, model learning rate α, matrix learning rate γ.
Output: Model with θ_{best}

1 Initialize T to an identity matrix
2 **for** *epoch* $e \in [1, \cdots, E]$ **do**
3 shuffle D_η and D_v
4 **for** $t \in [1, \cdots, \lceil N/n \rceil]$ **do**
5 $\{X_\eta, Y_\eta\} \leftarrow$ the tth batch of the training set
6 //Optimize the model parameters
7 $l_\eta \leftarrow J_c(X_\eta, Y_\eta; \theta^{(t)}, \mathsf{T}^{(t)})$
8 $\nabla_{\theta^{(t)}} l_\eta \leftarrow \text{Backward}(l_\eta, \theta^{(t)})$
9 $\theta^{(t+1)} \leftarrow \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} l_\eta$
10 //Optimize the noise transition matrix
11 $\{X_v, Y_v\} \leftarrow$ sample one batch from the validation set
12 $l_v \leftarrow J(X_v, Y_v; \theta^{(t+1)})$
13 $\nabla_{\mathsf{T}^{(t)}} l_v \leftarrow \text{Backward}(l_v, \mathsf{T}^{(t)})$
14 $\tilde{\mathsf{T}}^{(t+1)} \leftarrow \mathsf{T}^{(t)} - \gamma \nabla_{\mathsf{T}^{(t)}} l_v$
15 $\mathsf{T}^{(t+1)} \leftarrow \text{Normalization}(\tilde{\mathsf{T}}^{(t+1)})$
16 evaluate the model on D_v and maintain the current best parameter values θ_{best}
17 **return** θ_{best}

C. Algorithmic Details

In this section, we provide the algorithmic details of our framework.

Derivative Computation. The computation of $\nabla_{\theta^{(t)}} l_\eta$, the derivative of the training loss with respect to $\theta^{(t)}$, is straightforward, since T can be regarded as a constant and

$$\nabla_{\theta^{(t)}} l_\eta = \frac{1}{n} \sum_{(\mathbf{x}, y) \in \{X_\eta, Y_\eta\}} \sum_{y^* \in \mathcal{Y}} \mathsf{T}_{yy^*}^{(t)} \times \frac{\partial l(\mathbf{x}, y^*; \theta^{(t)})}{\partial \theta^{(t)}}. \tag{5}$$

The impact of T on the validation loss is indirectly made through $\theta^{(t+1)}$. According to the chain rule, we have

$$\nabla_{\mathsf{T}^{(t)}} l_v = \frac{\partial l_v}{\partial \theta^{(t+1)}} \times \frac{\partial \theta^{(t+1)}}{\partial \mathsf{T}^{(t)}}. \tag{6}$$

The first part $\nabla_{\theta^{(t+1)}} l_v$ is the derivative of validation loss with respect to model parameters, which is a straightforward derivative computation. As for the second part, let $\{X_\eta^i, Y_\eta^i\} \subset \{X_\eta, Y_\eta\}$ denote all examples in the training batch with label i, for a certain element $\mathsf{T}_{ij}^{(t)}$, we have:

$$\frac{\partial\theta^{(t+1)}}{\partial \mathsf{T}_{ij}^{(t)}} = -\alpha \times \frac{1}{n} \times \sum_{\mathbf{x}\in X_\eta^i} \frac{\partial l(\mathbf{x}, j; \theta^{(t)})}{\partial \theta^{(t)}}. \tag{7}$$

Proof of Equation 7: Let $\{X_\eta^i, Y_\eta^i\} \subset \{X_\eta, Y_\eta\}$ denote all examples in the training batch with label i. To obtain the derivative of $\theta^{(t+1)}$ with respect to $\mathsf{T}_{ij}^{(t)}$, we should pick out all items that contain $\mathsf{T}_{ij}^{(t)}$ and others (denoted by $\boxdot$) can be regarded as constant:

$$\begin{aligned}\theta^{(t+1)} =& \theta^{(t)} - \alpha \times \frac{1}{n} \sum_{(\mathbf{x},y)\in\{X_\eta, Y_\eta\}} \sum_{y^*\in\mathcal{Y}} \mathsf{T}_{yy^*}^{(t)} \times \frac{\partial l(\mathbf{x}, y^*; \theta^{(t)})}{\partial \theta^{(t)}} \\ =& \theta^{(t)} - \alpha \times \frac{1}{n} (\sum_{\mathbf{x}\in X_\eta^i} \mathsf{T}_{ij}^{(t)} \times \frac{\partial l(\mathbf{x}, j; \theta^{(t)})}{\partial \theta^{(t)}} + \boxdot).\end{aligned}$$

Then we have

$$\frac{\partial\theta^{(t+1)}}{\partial \mathsf{T}_{ij}^{(t)}} = -\alpha \times \frac{1}{n} \times \sum_{\mathbf{x}\in X_\eta^i} \frac{\partial l(\mathbf{x}, j; \theta^{(t)})}{\partial \theta^{(t)}}.$$

Efficiency. Algorithm 1 requires to perform forward and backward propagation on both the training set and the validation set. The backward pass on the validation set is like backward on backward propagation to get the gradients of T, which is more time-consuming. This is because for each element T_{ij}, we have to pick out all examples with label i and compute the gradient after flipping to label j based on Formula 7. Totally, $C \times C$ times backward computations are needed.

Some strategies can be adopted to improve the efficiency. First, we observe that there is actually no need to update the noise transition matrix T in every iteration of each epoch. Instead, we can sample some training and validation examples to compute the derivations at the beginning of each epoch, choose a larger learning rate to update T, and keep T unchanged throughout subsequent iterations in this epoch. Infrequent updates of T can improve the efficiency of training and make it easier to converge. Second, we can leverage the automatic differentiation technique to efficiently compute the gradient of the validation loss l_v with respect to T. However, the implementation is not straightforward since the connection between T and $\theta^{(t+1)}$ will be automatically discarded after updating the model parameters from $\theta^{(t)}$ to $\theta^{(t+1)}$ and the chain from l_v to T will be broken. But we can still apply the automatic differentiation technique to the fast calculation of $\nabla_{\theta^{(t+1)}} l_v$ and $\nabla_{\mathsf{T}}\theta^{(t+1)}$, and directly multiply these two items to get $\nabla_{\mathsf{T}^{(t)}} l_v$. This way of computation is irrelevant to the number of classes and can be easily applied to any model, but requires us to save a copy of $\theta^{(t+1)}$ outside the model to not lose the connection between T and $\theta^{(t+1)}$.

V. NNAR Noise Setting

In reality, it will be more practical to assume that the probability of label noise depends on not only the true class but also the features. For example, the instances that near the decision boundary are more likely to be mislabelled. In this situation, the loss function J_c will be too coarse-grained, where the examples with the same observed label i are considered to have the same probability of being incorrectly labelled as another label j. Thus we propose a new loss function for the NNAR noise setting, and further discuss how to learn with NNAR noise in this section.

A. Loss Function for NNAR Noise

We extend our loss function in NAR noise setting to NNAR noise setting. With NNAR noise, label corruption not only depends on the true label y^* but also the feature vector $\mathbf{x}$. This can be interpreted as each example $\mathbf{x}$ has a different label corruption transition matrix $\mathsf{T}_{\mathbf{x}}^*$ and also a different recovering transition matrix $\mathsf{T}_{\mathbf{x}}$. One example only has one label, thus the matrix $\mathsf{T}_{\mathbf{x}}$ reduces to a vector $\mathsf{V}_{\mathbf{x}}$, which is of length C and the i-th element $\mathsf{V}_{\mathbf{x}}[i] = p(y^* = i|\mathbf{x}, y)$ is the probability that the true label is i given feature $\mathbf{x}$ and observed label y. At the same time, $\sum_i \mathsf{V}_{\mathbf{x}}[i] = 1$ holds.

Then given a training example $(\mathbf{x}, y)$, the NNAR-noise-robust loss should be $l_f(\mathbf{x}, y; \theta, \mathsf{V}_{\mathbf{x}}) = \sum_{y^*\in\mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times l(\mathbf{x}, y^*; \theta)$, which is the linear combination of the loss for each possible true label whose coefficients are based on their probabilities given the corresponding transition vector and also can be denoted as $l_f(y, h_\theta(\mathbf{x}); \mathsf{V}_{\mathbf{x}})$. Then, the loss function is represented as

$$J_f = \frac{1}{N} \sum_{(\mathbf{x},y)\in D_\eta} \sum_{y^*\in\mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times l(\mathbf{x}, y^*; \theta). \tag{8}$$

Similarly, we can prove that the minimization of J_f on noisy data is equivalent to the minimization of the traditional loss on clean data.

Theorem 2: J_f is an unbiased loss function for the NNAR noise setting, i.e.,

$$\mathbb{E}_{\mathbf{x},y} l_f(\mathbf{x}, y; \theta, \mathsf{V}_{\mathbf{x}}) = \mathbb{E}_{\mathbf{x},y^*} l(\mathbf{x}, y^*; \theta).$$

Therefore, we have

$$\text{argmin } \mathbb{E}_{\mathbf{x},y} l_f(\mathbf{x}, y; \theta, \mathsf{V}_{\mathbf{x}}) = \text{argmin } \mathbb{E}_{\mathbf{x},y^*} l(\mathbf{x}, y^*; \theta).$$

Proof of Theorem 2: For each example $(\mathbf{x}, y)$, by Theorem 1, we have

$$\begin{aligned}\mathbb{E}_{y^*} l(\mathbf{x}, y^*; \theta) =& \mathbb{E}_y \sum_{y^*\in\mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times l(\mathbf{x}, y^*; \theta) \\ =& \sum_{y^*\in\mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times l(\mathbf{x}, y^*; \theta)\end{aligned}$$

$$\begin{aligned}\text{then } \mathbb{E}_{\mathbf{x},y} l_f(\mathbf{x}, y; \theta, \mathsf{V}_{\mathbf{x}}) =& \mathbb{E}_{\mathbf{x},y} \sum_{y^*\in\mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times l(\mathbf{x}, y^*; \theta) \\ =& \int p(\mathbf{x}) \sum_{y^*\in\mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times l(\mathbf{x}, y^*; \theta) d\mathbf{x} \\ =& \int p(\mathbf{x}) \mathbb{E}_{y^*} l(\mathbf{x}, y^*; \theta) d\mathbf{x} \\ =& \mathbb{E}_{\mathbf{x},y^*} l(\mathbf{x}, y^*; \theta)\end{aligned}$$

B. Learning with NNAR Noise

Here, we also utilize the meta-learning framework to optimize the model parameter θ and the transition vector $\mathsf{V}_{\mathbf{x}}$ of each training example $(\mathbf{x}, y) \in D_\eta$ at the same time.

Optimization of θ. Given a batch $\{X_\eta, Y_\eta\} = \{(\mathbf{x}_i, y_i), i \in [1, n]\}$ of training examples, we take out their corresponding transition vectors and merge them into a $n \times C$-size matrix V_{X_η}. Then, we fix V_{X_η} and optimize the model parameters to minimize the training loss on the mini-batch

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} l_\eta = \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} J_f(X_\eta, Y_\eta; \theta^{(t)}, \mathsf{V}^{(t)}_{X_\eta}), \tag{9}$$

where α is the learning rate of the model, and we have

$$\nabla_{\theta^{(t)}} l_\eta = \frac{1}{n} \sum_{(\mathbf{x},y)\in\{X_\eta, Y_\eta\}} \sum_{y^* \in \mathcal{Y}} \mathsf{V}_{\mathbf{x}}[y^*] \times \frac{\partial l(\mathbf{x}, y^*; \theta^{(t)})}{\partial \theta^{(t)}}. \tag{10}$$

Optimization of $\mathsf{V}_{\mathbf{x}}$. After updating model parameters to $\theta^{(t+1)}$, we sample a batch $\{X_v, Y_v\} = \{(\mathbf{x}^v_j, y^v_j),\ j \in [1, m]\}$ from the validation set and optimize V_{X_η} based on the validation loss on the batch

$$\tilde{\mathsf{V}}^{(t+1)}_{X_\eta} = \mathsf{V}^{(t)}_{X_\eta} - \gamma \nabla_{\mathsf{V}^{(t)}_{X_\eta}} l_v = \mathsf{V}^{(t)}_{X_\eta} - \gamma \nabla_{\mathsf{V}^{(t)}_{X_\eta}} J(X_v, Y_v; \theta^{(t+1)}), \tag{11}$$

where γ is the learning rate of optimizing V. Concretely, there is

$$\nabla_{\mathsf{V}^{(t)}_{X_\eta}} l_v = \frac{\partial l_v}{\partial \theta^{(t+1)}} \times \frac{\partial \theta^{(t+1)}}{\partial \mathsf{V}^{(t)}_{X_\eta}}, \tag{12}$$

and based on Formula 9 and 10, for a certain element $\mathsf{V}^{(t)}_{\mathbf{x}}[y^*]$, we have

$$\frac{\partial \theta^{(t+1)}}{\partial \mathsf{V}^{(t)}_{\mathbf{x}}[y^*]} = -\alpha \times \frac{\partial l(\mathbf{x}, y^*; \theta^{(t)})}{\partial \theta^{(t)}}. \tag{13}$$

Note that $\tilde{\mathsf{V}}^{(t+1)}_{X_\eta}$ also should be normalized to make sure that the sum of each row is one.

$$\begin{aligned} \tilde{\mathsf{V}}^{(t+1)}_{\mathbf{x}}[y^*] &= max(\tilde{\mathsf{V}}^{(t+1)}_{\mathbf{x}}[y^*], 0) \\ \mathsf{V}^{(t+1)}_{\mathbf{x}} &= \frac{\tilde{\mathsf{V}}^{(t+1)}_{\mathbf{x}}}{\sum \tilde{\mathsf{V}}^{(t+1)}_{\mathbf{x}} + \delta(\tilde{\mathsf{V}}^{(t+1)}_{\mathbf{x}})} \end{aligned} \tag{14}$$

Hyper-parameter Reduction. In the NNAR noise setting, we need to learn a transition vector $\mathsf{V}_{\mathbf{x}}$ of length C for every training example that may have label noise. Without any prior knowledge, for a training set with N examples, the total number of hyperparameters to learn is $N \times C$. When N is large, learning these hyperparameters can be computationally expensive and often requires a large validation set.

To solve this problem, our idea is to identify training examples that are highly likely to have correct labels and skip learning hyper-parameters for them. To identify such examples, we train a model Φ_η on the dirty training data D_η with a relatively high learning rate and do early-stopping to avoid overfitting. We observed that the examples that Φ_η can predict correctly, denoting D^c_η, are almost with correct labels; the other examples, denoting D^d_η, are mostly the mislabelled examples or some "difficult" examples. Therefore, we can skip learning hyper-parameters for examples in D^c_η. This observation rests on the fact that ML models tend to learn patterns before memorizing noise [41]. If we train models with only a few iterations, the ML model is not able to "memorize" noise and is more likely to make correct predictions on those "easy" examples with no label noise.

To verify this, we inject 20% and 40% feature-dependent label noise into the training set of `CIFAR10` dataset [42] respectively and train Φ_η (details can be found in Section VII-A). We observe that only 0.6% training examples are mislabelled but correctly predicted by the model Φ_η with 20% NNAR noise, which account for 0.9% of all correctly predicted data and 3% of mislabelled data, When there is 40% NNAR noise, 1.2% training examples are mislabelled but correctly predicted by Φ_η, which account for 2% of all correctly predicted data and 3% of mislabelled data. That is to say, there is no need to

Algorithm 2: Framework of Learning with J_f

Input: training set D_η with N examples, validation set D_v, training batch size n, validation batch size m, model parameter θ, the number of epochs E, model learning rate α, vector learning rate γ.
Output: Model with θ_{best}

```
Φ_η ← the model training with D_η
D^d_η, D^c_η ← the training examples that cannot/can be correctly predicted by Φ_η
for each example (x, y) ∈ D^d_η do
    Initialize V_x to the one-hot representation of class y
for epoch e ∈ [1, ..., E] do
    shuffle D^d_η, D^c_η and D_v
    for t ∈ [1, ··· , ⌈|D^d_η|/n⌉] do
        {X_d, Y_d} ← the tth batch of the training set D^d_η
        //Optimize the model parameters
        l_η ← J_f(X_d, Y_d; θ^(t), V^(t)_{X_d})
        θ^(t+1) ← θ^(t) − α∇_{θ^(t)} l_η
        //Optimize the noise transition vectors
        {X_v, Y_v} ← sample one batch from the validation set
        l_v ← J(X_v, Y_v; θ^(t+1))
        Ṽ^(t+1)_{X_d} ← V^(t)_{X_d} − γ∇_{V^(t)_{X_d}} l_v
        V^(t+1)_{X_d} ← Normalization(Ṽ^(t+1)_{X_d})
    for t ∈ [⌈|D^d_η|/n⌉ + 1, ··· , ⌈N/n⌉] do
        {X_c, Y_c} ← one batch of the training set D^c_η
        //Optimize the model parameters
        l_η ← J(X_c, Y_c; θ^(t))
        θ^(t+1) ← θ^(t) − α∇_{θ^(t)} l_η
    evaluate the model on D_v and maintain the current best parameter values θ_best
Return θ_best
```

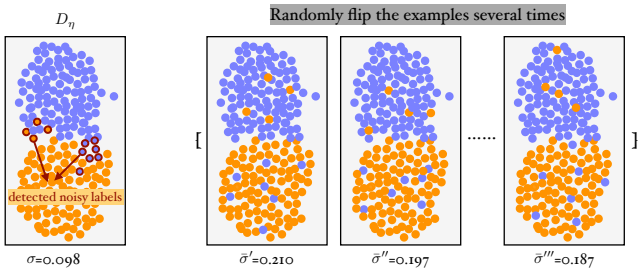


Fig. 4. Example of Noise Setting Detection

compute the transition vectors of the examples in D_η^c, whose labels can be basically determined to be correct. This pruning strategy can reduce 68.3% and 50.7% hyper-parameters to learn. This makes it possible to get rid of the dependence on large amounts of clean data, and it is also beneficial to the training efficiency since the time taken to train Φ_η is much less than to perform two full forward and backward propagation in each batch.

Framework Overview. Now it's time to give the whole workflow of learning with NNAR noise, which is shown in Algorithm 2. We utilize Φ_η to split the training examples into D_η^d and D_η^c (lines 1-2), and initialize the noise transition vector of each example $(\mathbf{x}, y)$ in D_η^d to the one-hot representation of class y (lines 3-4), indicating that no label noise is considered in the beginning; Then in each epoch, we first visit the training batch $\{X_d, Y_d\}$ of D_η^d, fix $\mathsf{V}_{X_d}^{(t)}$ to update the model parameters to $\theta^{(t+1)}$ and then update $\mathsf{V}_{X_d}^{(t)}$ to minimize the loss of sampled validation data (lines 7-16); Next, the training data in D_η^c is visited to further update the model parameters to minimize the traditional cross entropy loss (lines 17-21). We return the parameter θ_{best} which can achieve the best validation accuracy (line 16-17).

VI. Noise Setting Detection

Detecting the label noise setting of D_η is a necessary step to choose a proper loss function. Thus in this section, we present a novel method to detect the label noise distribution given a real-world dataset.

We observe that the noisy examples should be more evenly distributed in the data space when the probability of label error only depends on the true class, *i.e.,* T is "enough" to model the label noise. Let us start with the binary classification, a simpler case, to illustrate this observation. Given two classes $i, j \in \mathcal{Y}$ and two examples randomly picked from class j's actual sample space, *e.g.,* p_1, p_2 in Figure 1(b), in the NAR noise setting (the probability of label error is independent of examples' features), the number of data points incorrectly labelled as i around these two examples should be almost the same. On the contrary, in the NNAR noise setting, if one example is picked from the region where label error is prone to occur, *e.g.,* p_4 in Figure 1(c) and the other is not, *e.g.,* p_3 in Figure 1(c), the number of data points mislabelled as i around these two examples should vary a lot.

To employ this observation to detect noise settings, we need to know which data points are dirty. This is a "chicken-and-egg" problem. A common practice is to use the prediction from the pre-trained model Φ_η as pseudo true labels to detect dirty data. Admittedly, these dirty data points can deviate from the real dirty data points. We assume they roughly follow the same distribution as the real dirty data points so that we can use them as a substitute to detect noise settings. Since the model tends to make mistakes on data points around the decision boundary, the distribution of detected dirty data points can be biased toward NNAR, but this is acceptable as our J_f also gives relatively good performance in the NAR setting. In experiments, we will further validate that our method can find out the better loss function in most situations.

After obtaining the set of dirty data in D_η, we utilize the k-nearest neighbors algorithm to find the k neighbors closest to an example based on their features' distances. Then, the variance of the number of dirty neighbors can be computed to measure the degree of evenly mixing of the noisy and clean data in the data space. In NAR, the variance should be small, while in NNAR, the variance should be large. We first obtain the distribution of the variance under the null hypothesis and then compare the observed variance to this distribution to decide whether the null hypothesis should be rejected.

To simplify notation, we first consider a binary classification problem with positive/negative classes and we focus on data points in the positive class. Formally, let n_i denote the number of data points with noisy label in the k nearest neighbors of the ith data point in the positive class. We can obtain the averaged number of data points as $\mu = 1/N_p \sum_i n_i$ where N_p is the number of data points in the positive class and we can obtain the variance as $\sigma = 1/N_p \sum_i (n_i - \mu)^2$. Under the null hypothesis that noise labels are distributed evenly (formally, the probability of each data point having a wrong label is equal), the probability distribution of σ (i.e. $p(\sigma)$) is determined. Empirically, after detecting the label noise, we can compute the variance of the number of dirty neighbors $\hat{\sigma}$. We reject the null hypothesis when the observed variance $\hat{\sigma}$ is unlikely under the null hypothesis: $p(\sigma > \hat{\sigma}) < \alpha$ where α is a threshold commonly set as 0.05 [43].

The remaining challenge is that it is difficult to analytically derive the distribution $p(\sigma)$ as the result of k nearest neighbor search is dependent on the distribution of the data. Therefore, we propose to obtain the empirical distribution of $p(\sigma)$ through numerical simulation. Let $N_{p,\eta}$ denote the number of noisy labels in the positive class. The null hypothesis states that the probability of each data point having a wrong label is equal, so we can perform simulation by starting from clean labels and randomly choose $N_{p,\eta}$ data points to flip their labels. After flipping, we can obtain the variance of the number of dirty neighbors σ'. This is repeated many times and we get a list of variances $S = [\sigma'_1, \sigma'_2, \dots]$ which gives an empirical distribution $\hat{p}(\sigma)$. Note that k nearest neighbor search only needs to be done once because we can flip the labels on the result of k nearest neighbor search, so the simulation can be done efficiently. Similarly, we can perform the same procedure to do the hypothesis test on the negative class.

Figure 4 gives an example of noise setting detection. The

left shows the data space of the dirty training data, where four orange data points are detected as having noisy labels, *i.e.,* they should be purple. $\sigma = 0.098$ is the calculated variance of the number of dirty neighbors of each purple point. Then, we randomly flip four purple points to orange, repeat this process several times, and calculate the corresponding variance $\bar{\sigma}$. This is to simulate the distribution of the variance under the NAR noise setting. Suppose the range of σ' is [0.187,0.210], we can safely reject the null hypothesis (NAR setting) and conclude that D_η is in the NNAR noise setting.

The last to consider is how to deal with the multiple classification problem. Basically, we need to test all pairs of classes. Since those pairs with low error rate have insignificant noise distribution and little effect on accuracy, we actually focus on the variance of a few pairs of classes with the higher error rate. If the percentage of pairs of classes that determined to be NNAR is less than a pre-defined threshold (α can be used or a looser one to allow for the inaccuracy of error detection), we can safely conclude that the transition matrix in J_c is enough to model the label noise distribution.

VII. Experiments

We conduct experiments to answer the following questions: (Exp-1) How good is our approach under real-world label noise? (Exp-2) How good is our approach under injected class-dependent label noise? (Exp-3) How good is our approach under injected feature-dependent label noise? (Exp-4) What is the effect of noise setting detection module? (Exp-5) What is the impact of the size of clean validation set? (Exp-6) What is the effect of efficiency improvement strategies?

A. Experimental Setup

Datasets. We evaluate our approach on four publicly available datasets: `Food-101N` [39], `Clothing1M` [12], `CIFAR` [42] and `SST` [44]. The first three are image datasets, while the last one is a text dataset. The label noise in `Food-101N` and `Clothing1M` is from the real world, while the label noise in other datasets is artificially generated. The following are the details of these datasets.

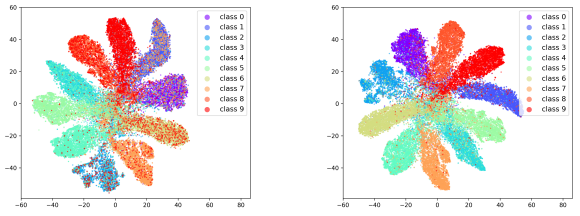
(1) *Food-101N* contains about 310K images of food recipes classified in 101 classes. The estimated label noise rate is about 20%. The dataset provides a verification label for each image that marks whether the corresponding class label is correct. We utilize the verification labels to build a clean validation set that contains 3K images, and directly adopt the test set of `Food-101` [45] for evaluation. Following previous work [39], we employ ImageNet pre-trained ResNet-50 model for this dataset.

(2) *Clothing1M* consists of more than 1 million images with noisy labels from 14 fashion classes. Images are obtained from the online shopping websites and labels are extracted from their surrounding texts. The estimated accuracy of class labels is about 61%. There exist additional validation and test sets with 14K and 10K examples whose labels are believed to be clean. Following previous work [12], we use the ImageNet pre-trained AlexNet model for this dataset.

(3) *CIFAR* contains two datasets of 32×32 color images: `CIFAR10` has 10 classes, with 6K images per class. There are 50K images for training and 10K for test. `CIFAR100` has 100 classes with 600 images per each. There are 500 training images and 100 testing images per class. We employ ImageNet pre-trained ResNet-50 for this dataset.

(4) *SST* (Stanford Sentiment Treebank) is a movie review dataset and we use the 2-class version, *i.e.,* `SST2`. It has 6,911 reviews in the training set, 872 in the validation set, and 1,821 in the test set. For this dataset, we use GloVe for word embedding and LSTM network for training.

Label Noise Generation. We inject either class-dependent or feature-dependent label noise into the training set of `CIFAR` and `SST` datasets under different noise rates 10%, 20%, 30% and 40%. For class-dependent noise, labels are corrupted according to the generated corruption transition matrix T^* where all diagonal elements are the correct rates of each class label (one minus the noise rates) and others are randomly filled such that the sum of each row is 1 [16], [31]. Then, the data with label i has probability T^*_{ij} to be changed to label j. For feature-dependent label noise, to the best of our knowledge, there exists no related work about how to inject it. Therefore, according to the common practice, we inject label noise to the examples near the decision boundary, which are less likely to belong to the current class. To do this, we train a model with clean data and obtain the predicted probabilities of each training example. Then we select the sample in ascending order of the likelihood of belonging to the current class and changed its current label to the first or second most likely one. The followings are examples of the distribution of our injected label noise. We utilize the trained ResNet-50 model to extract features of the images in `CIFAR10`, and t-SNE to reduce high-dimensional data to two dimensions for visualization. Although a lot of information is lost, we can still observe that the class-dependent label noise is randomly distributed in the data space. In contrast, the feature-dependent label noise is concentrated in a certain area.



(a) NAR Noise in CIFAR10 (b) NNAR Noise in CIFAR10

Fig. 5. Examples of Injected Label Noise Distribution

Baselines. We have implemented our approach in PyTorch[1]. For comparison, we implement (1) *Dirty*, the baseline model directly trained with the noisy training set in the cross entropy loss; For NAR noise setting, we obtain the open-source implementations of (2) *Cleanlab* [17], which estimates the noise transition matrix using the confidence learning technique and

[1]Code is released at https://github.com/Shuang-H/learning-with-noisy-label

Method	Food-101N	Clothing1M	CIFAR10		CIFAR100		SST	
			NAR (40%)	NNAR (40%)	NAR (40%)	NNAR (40%)	NAR (40%)	NNAR (40%)
Dirty	82.46	67.49	89.44	84.85	69.96	69.15	71.61	72.47
Cleanlab	83.77	61.76	91.96	82.96	67.67	67.07	71.07	72.47
LC	83.41	67.59	90.43	87.68	71.49	69.87	**73.42**	72.38
PENCIL	85.85	69.95	90.33	87.92	71.17	71.43	-	-
MW_Net	84.27	69.02	92.75	88.64	70.74	71.76	-	-
MLC	83.58	68.08	89.91	86.89	70.52	69.61	-	-
Ours	**86.15**	**71.82**	**93.17**	**91.19**	**75.28**	**72.28**	**73.42**	**73.74**

TABLE I
TEST ACCURACY ON VARIOUS DATASETS.

prune the examples with noisy labels before training. There are two training techniques to choose: directly training with the remaining "clean" data and co-teaching. We run both of them and report the best results. (3) *Loss Correction (LC)* [15], which approximates the transition matrix based on the predicted probabilities of noisy examples. LC has both forward and backward version which corrects the model prediction and the value of loss respectively. We run both of them and report the best results. For NNAR noise setting, we obtain the open-source implementations of (4) *PENCIL* [31], which introduces a probabilistic model updating both network parameters and label estimations. (5) *MW_Net* [40], which adaptively learns an explicit sample re-weighting function from the data in a meta-learning manner. The re-weighting function is formulated as an MLP network with one hidden layer. (6) *MLC* [36] is a meta-learning-based method that uses a meta-model to do label correction and a main model trained with the corrected labels.

Each competing method has its own hyper-parameters to decide. We use the same batch size to train the model and carefully tune other hyper-parameters via cross validation for all methods. For the fair comparison, all methods use the same validation set.

We have also compared with the baseline model Val trained with the clean validation set in the cross entropy loss. Even using pre-trained model, the accuracy of Val on CIFAR10 and CIFAR100 can only reach 80.89% and 27.75% respectively, which is much lower than other methods, thus we do not list it as a baseline here.

Clean Validation Set. All datasets have a validation set by default except CIFAR. As for CIFAR, we randomly sample 500 images from the training set for validation. We also change the random seed three times to sample different images into the validation set and report the average test accuracy we can achieve. For the fair comparison, we use the same validation set for all methods relying on it for training. If the compared approaches do not require the validation set, we use it to conduct an additional parameter tuning.

Experimental Environment. The evaluations on the real-world datasets are conducted in parallel on the server with 8 GeForce TITAN X GPU (The batch size is 128). Other evaluations are conducted on another server with one TITAN Xp GPU (The batch size is 16).

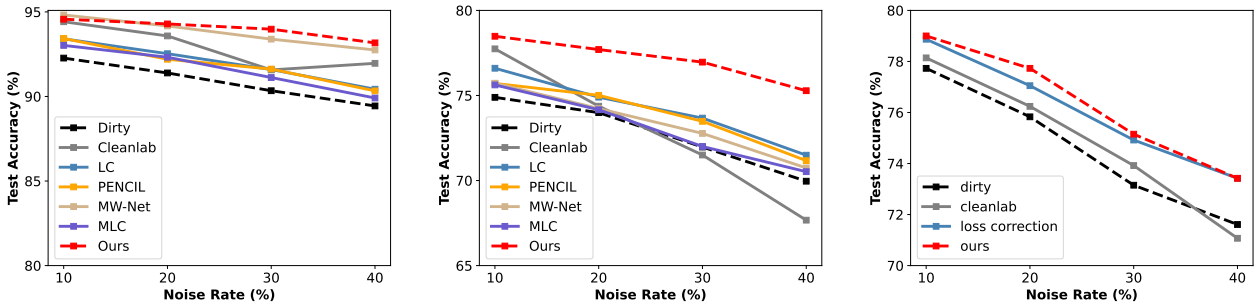
B. Experimental Results

Exp-1: Comparison under Real-world Noise. We evaluate our approach on Food-101N and Clothing1M to show the robustness of our loss function under real-world noises. Table I shows the accuracy of all methods. For all baselines, we give them a favor by running all of their possible settings (e.g., the forward and backward settings in loss correction) and report the best result. For our method, we have two settings: loss for NAR J_c and loss for NNAR J_f, and we use our proposed noise setting detection module to choose which one to use.

As shown in Table I, our method significantly outperforms other baselines. In Food-101N dataset, our test accuracy is 3.69% higher than directly learning from the dirty training set under the cross entropy loss, which confirms that our loss functions are robust to the real-world label noise. We perform better than LC since LC only uses the noise transition matrix to correct the loss, which cannot fully model the NNAR noise setting, while our approach also utilizes the noise transition vectors to capture the dependence between the feature and label noise. Our method also obtains higher test accuracy than PENCIL. This is because PENCIL only optimizes the loss on the training set, while our framework minimizes the corrected training loss and the validation loss at the same time. Compared with other meta-learning methods, our method is on average 2.1% better than MW_Net and 3.6% better than MLC. This is because MW_Net can loss useful information when assigning lower weights to examples, while our method corrects the loss without down-weighting any examples. MLC relies on training an extra network to learn how to correct labels. Therefore, its performance is not good when this network cannot generalize well. In comparison, our method learns to correct the loss directly without relying on any extra networks.

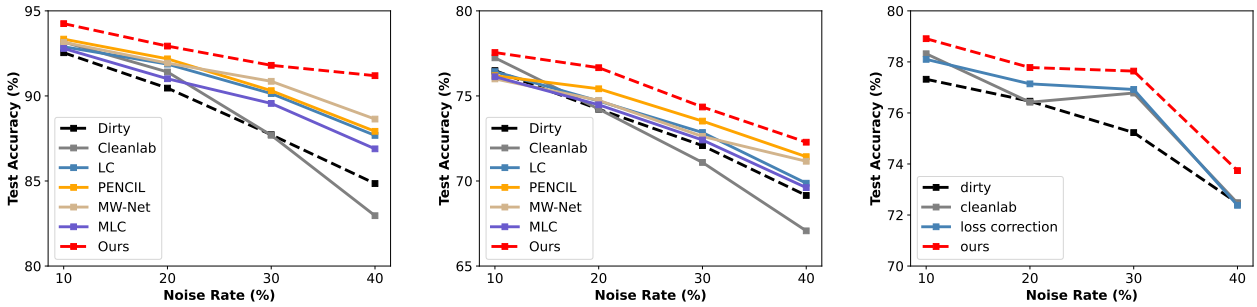
The results of Clothing1M dataset are similar. Note that we achieve 10.06% higher test accuracy than Cleanlab, which even performs worse than directly training with the dirty data. The reason is that Cleanlab removes a substantial amount of training examples that are detected as label-noisy data, which may include many difficult examples or rare examples in the minority classes that are important to the test accuracy.

Exp-2: Comparison under Injected NAR Noise. We inject the class-dependent label noise into CIFAR10, CIFAR100 and SST with different noise ratios 10%, 20%, 30%, 40% and



(a) CIFAR10 (b) CIFAR100 (c) SST

Fig. 6. Comparison under Different NAR Noise Rates



(a) CIFAR10 (b) CIFAR100 (c) SST

Fig. 7. Comparison under Different NNAR Noise Rates

the experimental results are shown in Table I and Figure 6. For clear presentation, Table I only shows the experimental results under 40% noise rate. Since the source codes of PENCIL, MW_Net and MLC work only for image datasets, its experimental results on SST are omitted.

From Table I, we can see that our method achieves higher test accuracy than all baselines in all datasets (LC can also achieve the highest test accuracy on SST dataset with 40% NAR noise). Particularly, our test accuracy is 3.73% higher than directly learning from the dirty training set in CIFAR10 and 5.32% higher in CIFAR100, showing the effectiveness of our loss functions under the NAR noise setting.

Figure 6 provides more details under different noise rates: our method performs best in almost all situations except in CIFAR10 with 10% NAR label noise, and with the increase of noise rate, the gap between our method and other baselines keep widening in most cases. It is worth noting that when the noise rate is relatively low, the test accuracy of Cleanlab in CIFAR is not much different from our method. The reason is that the model has been pre-trained with ImageNet dataset which basically can cover all classes of CIFAR. Even if Cleanlab may prune lots of training images by mistake, the impact on test accuracy is not great. But when the noise rate increases, Cleanlab even performs worse than directly training with the dirty data. This shows that in this situation, just deleting the noisy training examples is not a good choice.

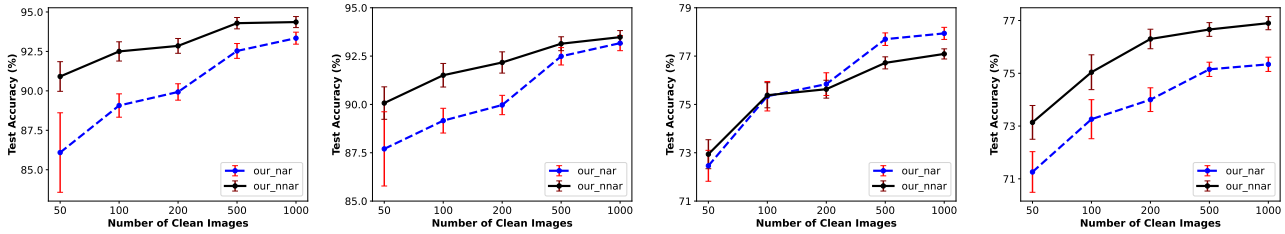
Exp-3: Comparison under Injected NNAR Noise. In this group of experiments, we inject the feature-dependent label noise into CIFAR and SST with different noise ratios. Table I shows the results under 40% NNAR noise and Figure 7 provides more details. Also, the results of PENCIL, MW_Net and MLC in SST dataset are omitted.

From Table I, we can see that in many cases, the accuracy achieved under feature-dependent label noise is lower than that under class-dependent label noise with the same error rate. This means that the feature-dependent label noise is more difficult to be handled. In this case, our method performs better than all baselines in all datasets, showing the effectiveness of our loss functions under the NNAR noise setting.

Figure 7 shows that our method significantly outperforms all other methods in all situations. With 40% NNAR noise, our test accuracy is 6.34% higher than directly learning with the dirty data and 3.27% higher than the best baseline in CIFAR10. The reason is that the feature-dependent label noise is relatively under-represented by the noise transition matrix in Cleanlab and LC no matter how it is learned from prior knowledge [15], [1] or directly from the data [35], [17]. This is also the reason why PENCIL, MW_Net and MLC perform better than Cleanlab and LC in most cases for NNAR noise setting.

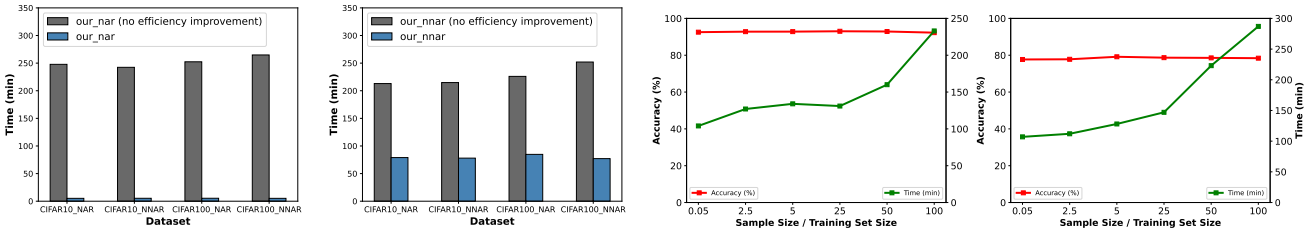
Exp-4: The Effect of Noise Setting Detection. In this part of experiment, we evaluate the effectiveness of our noise setting detection module. We inject class-dependent and feature-dependent label noise into CIFAR10 and CIFAR100 under different noise ratios 10%, 20%, 30%, 40%, and train both Φ_c and Φ_f to test whether our noise setting detection module can find the better loss function from J_c and J_f in each situation.

We observe that when J_c works better, the probability that our module chooses J_c as the final loss function is 100%; when J_f works better, the probability that our module chooses J_f as the final loss function is 92.86%. The results validate our



(a) CIFAR10 with NAR Noise (b) CIFAR10 with NNAR Noise (c) CIFAR100 with NAR Noise (d) CIFAR100 with NNAR Noise

Fig. 8. Effect of the Size of Clean Validation Set



(a) Time of Learning with J_c (an epoch)(b) Time of Learning with J_f (an epoch) (c) Efficiency v.s. Accuracy (CIFAR10) (d) Efficiency v.s. Accuracy (CIFAR100)

Fig. 9. Effect of Efficiency Improvement Strategies

statement that in most situations our method can pick out the better loss function.

Exp-5: Impact of the Size of Clean Validation Set. The transition matrix in J_c and transition vectors in J_f are optimized based on the loss on the clean validation set. Thus, in this group of experiments, we fix the size of the training set and varies the number of clean validation examples to explore the impact of the size of validation set on the final classification performance. The experimental results are shown in Figure 8. Note that we change the random seed three times to sample different images into the validation set. Here, the top of each bar is the highest test accuracy that can be reached when changing the random seed, and the bottom is the lowest value. We use lines to connect the center of each bar to show the trend of the test accuracy as the size of the validation set increases, and the height of the bar indicates the variance of the accuracy.

Figure 8 show that the increase of the size of clean validation set is beneficial to the classification performance. For example, when the number of clean examples increases from 50 to 1,000 in `CIFAR10` with 20% NAR label noise, the test accuracy increases from 86.09% to 93.34% using J_c and 90.91% to 94.36% using J_f. Therefore, our method may not work well if it is difficult to obtain such a validation set or if the clean validation set is very small. However, the test accuracy does not increase a lot after the size of validation set reaches 500. This suggests that we do not need to annotate a large number of clean data to guide the training process. Meanwhile, we can see that as the size of the validation set increases, the variance is getting smaller, which means that the choice of images in the validation set has a smaller and smaller impact on the test accuracy.

Exp-6: The Effect of Efficiency Improvement Strategies. In this group of experiments, we evaluate our efficiency improvement strategies and the results are shown in Figure 9. We pick four datasets: `CIFAR10` with 20% NAR noise, `CIFAR10` with 20% NNAR noise, `CIFAR100` with 20% NAR noise and `CIFAR100` with 20% NNAR noise. Overall, our strategies can greatly improve the learning efficiency in all situations. Figure 9(a) show that our sampling strategy proposed in Section IV can reduce the running time of an epoch from $\sim$250 min to $\sim$5 min; Figure 9(b) show that our hyper-parameter reduction strategy proposed in Section V can reduce the average time of an epoch from $\sim$226 min to $\sim$79 min.

We verify that our efficiency improvement strategy that uses a sample to update the transition matrix T does not hurt the accuracy of the model. Figure 9(c) and Figure 9(d) show the running time and accuracy of our model on CIFAR10 and CIFAR100 under different sample sizes. As the sample size decreases, the running time decreases significantly up to 60%, while the accuracy almost remains unchanged.

VIII. Conclusion

In this paper, we have proposed a model-agnostic approach for directly learning with noisy labels. We have presented two noise-robust loss functions, which are for different noise distributions. We have utilized the meta-learning technique to optimize the hyper-parameters in the loss functions under the supervision of the loss on the small but clean validation set. We have also designed a module to detect the label noise setting on hand for choosing a proper loss function, and proposed several strategies to improve the efficiency of learning. We have demonstrated the effectiveness of our approach on real-world label noise and artificially injected label noise.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (61902017), China Postdoctoral Science Foundation (2019M650468) and China Scholarship Council under the Grant No. 201907095032.

REFERENCES

[1] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, "Using trusted data to train deep networks on labels corrupted by severe noise," in *Advances in neural information processing systems*, 2018, pp. 10 456–10 465.

[2] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in neural information processing systems*, 2018, pp. 8778–8788.

[3] D. F. Nettleton, A. Orriols-Puig, and A. Fornells, "A study of the effect of different types of noise on the precision of supervised learning techniques," *Artificial intelligence review*, vol. 33, no. 4, pp. 275–306, 2010.

[4] L. P. Garcia, A. C. de Carvalho, and A. C. Lorena, "Effect of label noise in the complexity of classification problems," *Neurocomputing*, vol. 160, pp. 108–119, 2015.

[5] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *Advances in neural information processing systems*, 2018, pp. 8527–8537.

[6] P. Chen, B. B. Liao, G. Chen, and S. Zhang, "Understanding and utilizing deep neural networks trained with noisy labels," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1062–1070.

[7] H. Song, M. Kim, D. Park, Y. Shin, and J.-G. Lee, "Learning from noisy labels with deep neural networks: A survey," *arXiv preprint arXiv:2007.08199*, 2020.

[8] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[9] B. Frénay and M. Verleysen, "Classification in the presence of label noise: a survey," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 5, pp. 845–869, 2013.

[10] C. E. Brodley and M. A. Friedl, "Identifying mislabeled training data," *Journal of Artificial Intelligence Research*, vol. 11, pp. 131–167, 1999.

[11] S. Lallich, F. Muhlenbach, and D. A. Zighed, "Improving classification by removing or relabeling mislabeled instances," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 2002, pp. 5–15.

[12] T. Xiao, T. Xia, Y. Yang, C. Huang, and X. Wang, "Learning from massive noisy labeled data for image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2691–2699.

[13] A. Vahdat, "Toward robustness against label noise in training deep discriminative neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 5596–5605.

[14] J. Yao, J. Wang, I. W. Tsang, Y. Zhang, J. Sun, C. Zhang, and R. Zhang, "Deep learning from noisy image labels with quality embedding," *IEEE Transactions on Image Processing*, vol. 28, no. 4, pp. 1909–1922, 2018.

[15] G. Patrini, A. Rozza, A. Krishna Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: A loss correction approach," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1944–1952.

[16] A. Ghosh, H. Kumar, and P. Sastry, "Robust loss functions under label noise for deep neural networks," pp. 1919–1925, 2017.

[17] C. G. Northcutt, L. Jiang, and I. L. Chuang, "Confident learning: Estimating uncertainty in dataset labels," *arXiv preprint arXiv:1911.00068*, 2019.

[18] S. Sukhbaatar, J. B. Estrach, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[19] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[20] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619–1630, 2006.

[21] P. Melville, N. Shah, L. Mihalkova, and R. J. Mooney, "Experiments on ensembles with missing and noisy data," in *International Workshop on Multiple Classifier Systems*. Springer, 2004, pp. 293–302.

[22] S. Azadi, J. Feng, S. Jegelka, and T. Darrell, "Auxiliary image regularization for deep cnns with noisy labels," *arXiv preprint arXiv:1511.07069*, 2015.

[23] I. Jindal, M. Nokleby, and X. Chen, "Learning deep networks from noisy labels with dropout regularization," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 967–972.

[24] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, "Convexity, classification, and risk bounds," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 138–156, 2006.

[25] Y. Xia, X. Cao, F. Wen, G. Hua, and J. Sun, "Learning discriminative reconstructions for unsupervised outlier removal," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1511–1519.

[26] W. Liu, G. Hua, and J. R. Smith, "Unsupervised one-class learning for automatic outlier removal," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3826–3833.

[27] X. Liu, S. Li, M. Kan, S. Shan, and X. Chen, "Self-error-correcting convolutional neural network for learning with noisy labels," in *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. IEEE, 2017, pp. 111–117.

[28] N. Matic, I. Guyon, L. Bottou, J. Denker, and V. Vapnik, "Computer aided cleaning of large databases for character recognition," in *11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, 1992, pp. 330–331.

[29] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.

[30] D. Zhang, D. Meng, and J. Han, "Co-saliency detection via a self-paced multiple-instance learning framework," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 5, pp. 865–878, 2016.

[31] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," *arXiv preprint arXiv:1803.09050*, 2018.

[32] Y. Wang, W. Liu, X. Ma, J. Bailey, H. Zha, L. Song, and S.-T. Xia, "Iterative learning with open-set noisy labels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8688–8696.

[33] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *International Conference on Machine Learning*, 2018, pp. 2304–2313.

[34] A. Ghosh, N. Manwani, and P. Sastry, "Making risk minimization tolerant to label noise," *Neurocomputing*, vol. 160, pp. 93–107, 2015.

[35] Z. Wang, G. Hu, and Q. Hu, "Training noise-robust deep neural networks via meta-learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 4524–4533.

[36] G. Zheng, A. H. Awadallah, and S. Dumais, "Meta label correction for noisy label learning," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

[37] S. Sukhbaatar and R. Fergus, "Learning from noisy labels with deep neural networks," *arXiv preprint arXiv:1406.2080*, vol. 2, no. 3, p. 4, 2014.

[38] A. J. Bekker and J. Goldberger, "Training deep neural-networks based on unreliable labels," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 2682–2686.

[39] K.-H. Lee, X. He, L. Zhang, and L. Yang, "Cleannet: Transfer learning for scalable image classifier training with label noise," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5447–5456.

[40] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, "Meta-weight-net: learning an explicit mapping for sample weighting," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, pp. 1919–1930.

[41] D. Arpit, S. Jastrzebski, 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," *arXiv preprint arXiv:1706.05394*, 2017.

[42] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[43] J. Neyman, "The emergence of mathematical statistics," *On the history of statistics and probability*, pp. 68–121, 1976.

[44] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[45] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101–mining discriminative components with random forests," in *European conference on computer vision*. Springer, 2014, pp. 446–461.