Generalized Data Weighting via Class-Level Gradient Manipulation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Label noise and class imbalance are two major issues coexisting in real-world datasets. To alleviate the two issues, state-of-the-art methods reweight each instance by leveraging a small amount of clean and unbiased data. Yet, these methods overlook class-level information within each instance, which can be further utilized to improve performance. To this end, in this paper, we propose **G**eneralized **D**ata **W**eighting (**GDW**) to simultaneously mitigate label noise and class imbalance by manipulating gradients at the class level. To be specific, GDW unrolls the loss gradient to class-level gradients by the chain rule and reweights the flow of each gradient separately. In this way, GDW achieves remarkable performance improvement on both issues. Aside from the performance gain, GDW efficiently obtains class-level weights **without** introducing any extra computational cost compared with instance weighting methods. Specifically, GDW performs a gradient descent step on class-level weights, which only needs intermediate gradients. Extensive experiments in various settings verify the effectiveness of GDW. For example, GDW outperforms state-of-the-art methods by 2.56% under the 60% uniform noise setting in CIFAR10. Our code will be available upon acceptance.

1 Introduction

Real-world classification datasets often suffer from two issues, i.e., label noise [1] and class imbalance [2]. On the one hand, label noise often results from the limitation of data generation, e.g., sensor errors [3] and mislabeling from crowdsourcing workers [4]. Label noise misleads the training process of DNNs and degrades the model performance in various aspects [5, 6, 7]. On the other hand, imbalanced datasets are either naturally long-tailed [8, 9] or biased from the real-world distribution due to imperfect data collection [10, 11]. Training with imbalanced datasets usually results in poor classification performance on weakly represented classes [12, 13, 14]. Even worse, these two issues often coexist in the real-world datasets [15].

To prevent the model from memorizing noisy information, many important work have been proposed, including label smoothing [16], noise adaptation [17], importance weighting [18], GLC [19], and Co-teach [20]. Meanwhile, [12, 13, 14, 21] propose effective methods to tackle class imbalance. However, these methods inevitably introduce hyper-parameters (e.g., the weighting factor in [13] and the focusing parameter in [21]), raising the difficulty in the real-world deployment.

Inspired by recent advances in meta-learning, some work [22, 23, 24, 25] propose to solve both issues by leveraging a clean and unbiased meta set. These methods treat instance weights as hyper-parameters and dynamically update these weights to circumvent hyper-parameter tuning. Specifically, MWNet [23] adopts a MLP with the instance loss as input and the instance weight as output. Due to the MLP, MWNet has better scalability on large datasets compared with INSW [24]. Although these methods can handle label noise and class imbalance to some extent, they cannot fully utilize class-level information within each instance, resulting in the potential loss of useful information. For example, in a three-class classification task, every instance has three logits. As shown in Figure 1, every logit corresponds to a class-level gradient flow due to the existence of the loss function. These gradient flows represent three kinds of information: "not cat", "dog", and "not bird". Instance weighting methods [23, 22] alleviate label noise by downweighting all the gradient flows of the instance, which discards three kinds of information simultaneously. Yet, downweighting the "not bird" gradient flow is a waste of information. Similarly, in class imbalance scenarios, different gradient flows represent different class-level information. Therefore, it is necessary to reweight instances at the class level for better information use.

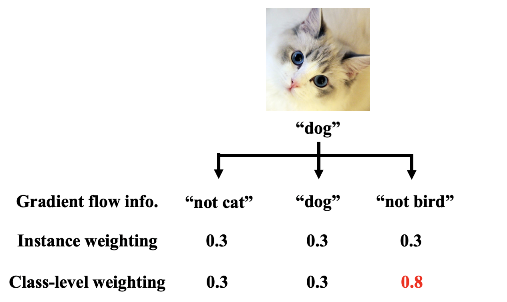


Figure 1: Motivation for class-level weighting. For a noisy instance (e.g. cat mislabeled as "dog"), all gradient flows are downweighted by instance weighting. Although the gradient flows for "dog" and "not cat" contain harmful information, the gradient flow for "not bird" is still valuable for training, which should not be downweighted.

To this end, we propose Generalized Data Weighting (**GDW**) to tackle label noise and class imbalance by class-level gradient manipulation. Firstly, we introduce class-level weights to represent the importance of different gradient flows and then propose class-level weighting by gradient manipulation. Secondly, we impose a zero-mean constraint on class-level weights for stable training. Thirdly, to efficiently obtain class-level weights, we develop a two-stage weight generation scheme embedded in the bi-level optimization. Instance weighting methods [22, 23, 24, 25] are special cases of GDW when class-level weights within any instance are the same. In this way, GDW achieves impressive performance improvement in various settings.

To sum up, our contribution is two-fold:

1. For better information utilization, we propose GDW, a generalized data weighting method, which better handles label noise and class imbalance. To the best of our knowledge, we are the first to propose class-level weighting on gradient flows.
2. To obtain class-level weights efficiently, we design a two-stage scheme embedded in the bi-level optimization framework, which does not introduce any extra computational cost. To be specific, during the back-propagation we store intermediate gradients, with which we update class-level weights via a gradient descent step.

2 Related work

2.1 Traditional Methods for Label Noise

Label noise is a common problem in classification tasks [5, 6, 7]. To avoid overfitting to label noise, [16] propose label smoothing to regularize the model. [17, 26] form different models to indicate the relation between noisy instances and clean instances. [18] estimate an importance weight for each instance to represent its value to the model. [20] train two models simultaneously and let them teach each other in every mini-batch. However, without a clean dataset, these methods cannot handle severe noise [22]. [19] correct the prediction of the model by estimating the label corruption matrix via a clean validation set, but this matrix is the same across all instances. Instead, our method generates dynamic class-level weights for every instance to improve training.

2.2 Traditional Methods for Class Imbalance

Many important work have been proposed to handle class imbalance [27, 28, 29, 21, 13, 12, 14, 30]. [27, 30] propose to over-sample the minority class and under-sample the majority class. [28, 29] learn a class-dependent cost matrix to obtain robust representations for both majority and minority classes. [21, 13, 12, 14] design a reweighting scheme to rebalance the loss for each class. These methods are quite effective, whereas they need to manually choose loss functions or hyper-parameters. In contrast, meta-learning methods view instance weights as hyper-parameters and dynamically update them via a meta set to avoid hyper-parameter tuning.

Table 1: Related work comparison. "Noise" and "Imbalance" denote whether the method can solve label noise and class imbalance. "Class-level" denotes whether the method utilizes class-level information in each instance, and "Scalability" denotes whether the method can scale to large datasets.

	Focal [21]	Balanced [13]	Co-teach [20]	GLC [19]	L2RW [22]	INSW [24]	MWNet [23]	Soft-label [39]	Gen-label [37]	**GDW**
Noise	×	×	✓	✓	✓	✓	✓	✓	✓	✓
Imbalance	✓	✓	×	×	✓	✓	✓	×	×	✓
Class-level	×	×	×	×	×	×	×	✓	✓	✓
Scalability	✓	✓	✓	✓	✓	×	✓	×	×	✓

2.3 Meta-Learning Methods

With recent development in meta-learning [31, 32, 33], many important methods have been proposed to handle label noise and class imbalance via a meta set [34, 35, 36, 23, 22, 25, 24, 37]. [36] propose MentorNet to provide a data-driven curriculum for the base network to focus on correct instances. To distill effective supervision, [38] estimate pseudo labels for noisy instances with a meta set. To provide dynamic regularization, [39, 37] treat labels as learnable parameters and adapt them to the model's state. Although these methods can tackle label noise, they introduce huge amounts of learnable parameters and thus cannot scale to large datasets. To alleviate class imbalance, [34] describe a method to learn from long-tailed datasets. Specifically, [34] propose to encode meta-knowledge into a meta-network and model the tail classes by transfer learning.

Furthermore, many meta-learning methods propose to mitigate the two issues by reweighting every instance [23, 22, 25, 40, 24]. [40] equip each instance and each class with a learnable parameter to govern their importance. By leveraging a meta set, [23, 22, 25, 24] learn instance weights and model parameters via bi-level optimization to tackle label noise and class imbalance. [22] assign weights to training instances only based on their gradient directions. Furthermore, [24] combine reinforce learning and meta-learning, and treats instance weights as rewards for optimization. Meanwhile, [23, 25] adopt a weighting network to output weights for instances and use bi-level optimization to jointly update the weighting network parameters and model parameters. Although these methods handle label noise and class imbalance by reweighting instances, a scalar weight for every instance cannot capture class-level information, as shown in Figure 1. Therefore, we introduce class-level weights for different gradient flows and adjust them to better utilize class-level information.

We show the differences between GDW and other related methods in Table 1.

3 Method

3.1 Notations

In most classification tasks, there is a training set $D_{train} = \{(x_i, y_i)\}_{i=1}^{N}$ and a meta set $D_{meta} = \{(x_i^v, y_i^v)\}_{i=1}^{M}$. We aim to alleviate label noise and class imbalance in the D_{train} with the clean unbiased D_{meta}. The model parameters are denoted as $\boldsymbol{\theta}$, and the number of classes is denoted as C.

3.2 Class-level Weighting by Gradient Manipulation

To utilize class-level information, we learn a class-level weight for every gradient flow instead of a scalar weight for all C gradient flows in [23]. Denote $\mathcal{L}$ as the loss of any instance. Applying the chain rule, we unroll the gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{\theta}$ as

$$\nabla_{\boldsymbol{\theta}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{L}}{\partial \mathbf{l}} \frac{\partial \mathbf{l}}{\partial \boldsymbol{\theta}} \doteq \mathbf{D}_1 \mathbf{D}_2, \tag{1}$$

where $\mathbf{l} \in \mathbb{R}^C$ represents the predicted logit vector of the instance. We introduce class-level weights $\boldsymbol{\omega} \in \mathbb{R}^C$ and denote the j^{th} component of $\boldsymbol{\omega}$ as the normal-font ω_j. To indicate the importance of every gradient flow, we perform an element-wise product $f_{\boldsymbol{\omega}}(\cdot)$ on $\mathbf{D}_1$ with $\boldsymbol{\omega}$. After this manipulation, the gradient becomes

$$f_{\boldsymbol{\omega}}\left(\nabla_{\boldsymbol{\theta}}\mathcal{L}\right) \doteq \left(\boldsymbol{\omega} \otimes \frac{\partial \mathcal{L}}{\partial \mathbf{l}}\right) \frac{\partial \mathbf{l}}{\partial \boldsymbol{\theta}} = \left(\boldsymbol{\omega} \otimes \mathbf{D}_1\right) \mathbf{D}_2 \doteq \mathbf{D}_1' \mathbf{D}_2, \tag{2}$$

where $\otimes$ denotes the element-wise product of two vectors. Note that ω_j represents the importance of the j^{th} gradient flow. Obviously, instance weighting is a special case of GDW when elements of $\boldsymbol{\omega}$

are the same. Most classification tasks [41, 42, 43] adopt the *Softmax-CrossEntropy* loss. In this case, we have $\mathbf{D}_1 = \mathbf{p} - \mathbf{y}$, where $\mathbf{p} \in \mathbb{R}^C$ denotes the probability vector output by *softmax* and $\mathbf{y} \in \mathbb{R}^C$ denotes the one-hot label of the instance.

As shown in Figure 1, for a noisy instance (e.g., cat mislabeled as "dog"), instance weighting methods assign a low scalar weight to all gradient flows of the instance. Instead, GDW assigns class-level weights to different gradient flows by leveraging the meta set. Specifically, GDW tries to downweight the gradient flows for "dog" and "not cat", and upweight the gradient flow for "not bird". Similarly, in imbalance settings, different gradient flows have different class-level information. Thus GDW can also better handle class imbalance by adjusting the importance of different gradient flows.

3.3 Zero-mean Constraint on Class-level Weights

To retain the *Softmax-CrossEntropy* loss structure after the manipulation, we impose a zero-mean constraint on $\mathbf{D}_1'$. To be specific, we analyze the j^{th} element of $\mathbf{D}_1'$ (see Appendix A for details):

$$\omega_j(p_j - \mathbf{y}_j) = \omega_t\left(p_j' - \mathbf{y}_j\right) + \left(\sum_k \omega_k p_k - \omega_t\right) p_j'. \tag{3}$$

where $p_j' \doteq \frac{\omega_j p_j}{\sum_k \omega_k p_k}$ is the weighted probability, and ω_t denotes the class-level weight at the target (label) position. We observe that the first term in Eq. 3 satisfies the structure of the gradient of the *Softmax-CrossEntropy* loss, and thus propose to eliminate the second term which messes the structure. Specifically, we let

$$\sum_k \omega_k p_k - \omega_t = 0 \Rightarrow \omega_t = \frac{\sum_{j \neq l} \omega_j p_j}{1 - p_t}, \tag{4}$$

where p_t is the probability of the target class. Note that $\sum_j \omega_j \mathbf{y}_j = \omega_t$, and thus we have

$$\sum_j \omega_j(p_j - \mathbf{y}_j) = 0. \tag{5}$$

This restricts the mean of $\mathbf{D}_1'$ to be zero. Therefore, we name this constraint as **zero-mean constraint**. With this, we have

$$\mathbf{D}_1' = \omega_t\left(\mathbf{p}' - \mathbf{y}\right). \tag{6}$$

Eq. 6 indicates that $\boldsymbol{\omega}$ adjust the gradients in two levels, i.e., instance level and class level. To be specific, ω_t acts as the instance-level weight in previous instance weighting methods [23, 22, 24, 25]. Class-level weights manipulate gradient flows by adjusting the probability from $\mathbf{p}$ to $\mathbf{p}'$.

3.4 Efficient Two-stage Weight Generation Embedded in Bi-level Optimization

In this subsection, we first illustrate the three-step bi-level optimization framework in [23]. Furthermore, we embed a two-stage scheme in the bi-level optimization framework to efficiently obtain class-level weights, with which we manipulate gradient flows and optimize model parameters.

Three-step Bi-level Optimization. Generally, the goal of classification tasks is to obtain the optimal model parameters $\boldsymbol{\theta}^*$ by minimizing the average loss on D_{train}, denoted as $\frac{1}{N}\sum_{i=1}^{N} l_{train}(x_i, y_i; \boldsymbol{\theta})$. As an instance weighting method, [23] adopt a three-layer MLP parameterized by $\boldsymbol{\phi}$ as the weighting network and take the loss of the i^{th} instance as input and output a scalar weight ω_i. Then $\boldsymbol{\theta}^*$ is optimized by minimizing the instance-level weighted training loss:

$$\boldsymbol{\theta}^*(\boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{N}\sum_{i=1}^{N} \omega_i(\boldsymbol{\phi}) l_{train}(x_i, y_i; \boldsymbol{\theta}) \tag{7}$$

To obtain the optimal ω_i, they propose to use a meta set as meta-knowledge and minimize the meta-loss to obtain $\boldsymbol{\phi}^*$:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}} \frac{1}{M}\sum_{i=1}^{M} l_{val}(x_i^v, y_i^v; \boldsymbol{\theta}^*(\boldsymbol{\phi})) \tag{8}$$

Since the optimization for $\boldsymbol{\theta}^*(\boldsymbol{\phi})$ and $\boldsymbol{\phi}^*$ is nested, they adopt an online strategy to update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ with a three-step optimization loop for efficiency. Denote the two sets of parameters at the τ^{th} loop as $\boldsymbol{\theta}_\tau$ and $\boldsymbol{\phi}_\tau$ respectively, and then the three-step loop is formulated as:

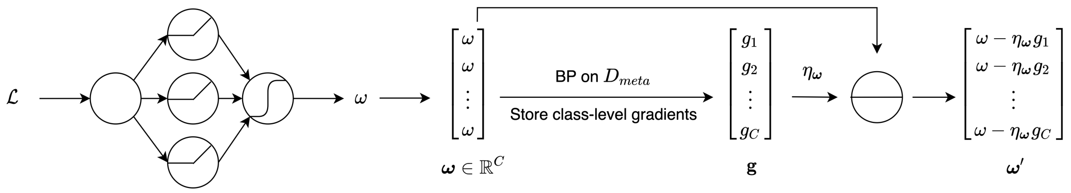


Figure 2: Two-stage Weight Generation. "BP" denotes the back-propagation in **Step 2** of the bi-level optimization framework. $\mathbf{g}$ denotes the intermediate gradients w.r.t. $\boldsymbol{\omega}$. $\ominus$ denotes the minus operator. Note that $\boldsymbol{\omega}$ is the first-stage (instance-level) weight and $\boldsymbol{\omega}'$ is the second-stage (class-level) weight.

Step 1 Update $\boldsymbol{\theta}_{\tau-1}$ to $\hat{\boldsymbol{\theta}}_\tau(\boldsymbol{\phi})$ via an SGD step on a mini-batch training set by Eq. 7.

Step 2 With $\hat{\boldsymbol{\theta}}_\tau(\boldsymbol{\phi})$, update $\boldsymbol{\phi}_{\tau-1}$ to $\boldsymbol{\phi}_\tau$ via an SGD step on a mini-batch meta set by Eq. 8.

Step 3 With $\boldsymbol{\phi}_\tau$, update $\boldsymbol{\theta}_{\tau-1}$ to $\boldsymbol{\theta}_\tau$ via an SGD step on the same mini-batch training set by Eq. 7.

Instance weights in **Step 3** are better than those in **Step 1**, and thus are used to update $\boldsymbol{\theta}_{\tau-1}$.

Two-stage Weight Generation. To guarantee scalability, we apply the same weighting network in [23] to obtain weights. To efficiently train $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, we also adopt the three-step bi-level optimization framework. Moreover, we propose an efficient two-stage scheme embedded in **Step 1-3** to generate class-level weights. This process does not introduce any extra computational cost compared to MWNet. We keep the notations of $\boldsymbol{\theta}_\tau$ and $\boldsymbol{\phi}_\tau$ unchanged.

The first stage is embedded in **Step 1**. Specifically, we obtain the first-stage class-level weights $\boldsymbol{\omega}_i = \omega_i \mathbf{1}$, by cloning the output of the weighting network for C times. Then we leverage the cloned weights $\boldsymbol{\omega}_i$ to manipulate gradients and update $\boldsymbol{\theta}$ with a mini-batch of training instances:

$$\hat{\boldsymbol{\theta}}_\tau\left(\boldsymbol{\phi}_{\tau-1}\right) \leftarrow \boldsymbol{\theta}_{\tau-1} - \eta_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} f_{\boldsymbol{\omega}_i(\boldsymbol{\phi}_{\tau-1})}\left(\nabla_{\boldsymbol{\theta}} l_{train}(x_i, y_i; \boldsymbol{\theta}_{\tau-1})\right) \tag{9}$$

where n is the mini-batch size, $\eta_{\boldsymbol{\theta}}$ is the learning rate of $\boldsymbol{\theta}$, and $f_{\boldsymbol{\omega}_i(\boldsymbol{\phi}_{\tau-1})}(\cdot)$ is the gradient manipulation operation defined in Eq. 2.

The second stage is embedded in **Step 2** and **Step 3**. Specifically in **Step 2**, GDW optimizes $\boldsymbol{\phi}$ with a mini-batch meta set:

$$\boldsymbol{\phi}_\tau \leftarrow \boldsymbol{\phi}_{\tau-1} - \eta_{\boldsymbol{\phi}} \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\phi}_{\tau-1}} l_{meta}(x_i^v, y_i^v; \hat{\boldsymbol{\theta}}_\tau(\boldsymbol{\phi}_{\tau-1})) \tag{10}$$

where m is the mini-batch size and $\eta_{\boldsymbol{\phi}}$ is the learning rate of $\boldsymbol{\phi}$. During the back-propagation in updating $\boldsymbol{\phi}_\tau$, GDW generates the second-stage weights using the intermediate gradients $\mathbf{g}_i$ on $\boldsymbol{\omega}_i$. To be specific,

$$\boldsymbol{\omega}_i' = \boldsymbol{\omega}_i - \eta_{\boldsymbol{\omega}} \mathbf{g}_i \tag{11}$$

Then we impose the zero-mean constraint proposed in Section 3.3 on $\boldsymbol{\omega}_i'$, which is later used in **Step 3** to update $\boldsymbol{\theta}_{\tau-1}$. Note that the two-stage weight generation scheme does not introduce any extra computational cost compared to MWNet because this generation process only utilizes the intermediate gradients during the back-propagation. In **Step 3**, we use $\boldsymbol{\omega}_i'$ to manipulate gradients and update the model parameters $\boldsymbol{\theta}_{\tau-1}$:

$$\boldsymbol{\theta}_\tau \leftarrow \boldsymbol{\theta}_{\tau-1} - \eta_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} f_{\boldsymbol{\omega}_i'}\left(\nabla_{\boldsymbol{\theta}} l_{train}(x_i, y_i; \boldsymbol{\theta}_{\tau-1})\right) \tag{12}$$

The only difference between **Step 1** and **Step 3** is that we use $\boldsymbol{\omega}_i'$ instead of the cloned output of the weighting network $\boldsymbol{\omega}_i$ to optimize $\boldsymbol{\theta}$. Since we only introduce $\boldsymbol{\phi}$ as extra learnable parameters, GDW can scale to large datasets. We summarize GDW in Algorithm 1. Moreover, we visualize the two-stage weight generation process in Figure 2 for better demonstration.

4 Experiments

We conduct extensive experiments on classification tasks to examine the performance of GDW. We compare GDW with other methods in the label noise setting and class imbalance setting in

Algorithm 1 Generalized Data Weighting via Class-Level Gradients Manipulation

```
    Input: Training set: $D_{train}$, Meta set: $D_{meta}$, batch size $n, m$, # of iterations $T$
    Initial model parameters: $\boldsymbol{\theta}_0$, initial weighting network parameters: $\boldsymbol{\phi}_0$
    Output: Trained model: $\boldsymbol{\theta}_T$
1 for $\tau \leftarrow 1$ to $T$ do
2     $\{x_i, y_i\}_{i=1}^{n} \leftarrow$ SampleFrom($D_{train}$)
3     $\{x_i^v, y_i^v\}_{i=1}^{m} \leftarrow$ SampleFrom($D_{meta}$)
4     Generate $\boldsymbol{\omega}_i$ from $\mathcal{L}_i$ via the weighting network parameterized by $\boldsymbol{\phi}_{\tau-1}$
5     Manipulate gradients by Eq. 2 and update $\hat{\boldsymbol{\theta}}_\tau$ by Eq. 9
6     Update $\boldsymbol{\phi}_\tau$ by Eq. 10;
7     Update $\boldsymbol{\omega}_i$ to $\boldsymbol{\omega}_i'$ by Eq. 11 and constrain $\boldsymbol{\omega}_i'$ by Eq. 4
8     Manipulate gradients with $\boldsymbol{\omega}_i'$ by Eq. 2 and update $\boldsymbol{\theta}_\tau$ by Eq. 12
```

Table 2: Test accuracy on CIFAR10 and CIFAR100 with different uniform noise ratios.

Dataset	CIFAR10			CIFAR100		
	0%	40%	60%	0%	40%	60%
BaseModel	92.73 ± 0.37	84.38 ± 0.32	77.92 ± 0.29	70.42 ± 0.54	57.28 ± 0.80	46.86 ± 1.54
Fine-tuning	92.77 ± 0.37	84.73 ± 0.47	78.41 ± 0.31	70.52 ± 0.57	57.38 ± 0.87	47.06 ± 1.47
Co-teach	91.54 ± 0.39	85.26 ± 0.56	78.90 ± 6.64	68.33 ± 0.13	$\underline{59.58 \pm 0.83}$	37.74 ± 2.60
GLC	90.85 ± 0.22	86.12 ± 0.54	$\underline{81.55 \pm 0.60}$	65.05 ± 0.59	56.99 ± 0.82	41.74 ± 1.98
L2RW	89.70 ± 0.50	84.66 ± 1.21	79.98 ± 1.18	63.40 ± 1.31	47.06 ± 4.84	36.02 ± 2.17
INSW	92.70 ± 0.57	84.88 ± 0.64	78.77 ± 0.82	70.52 ± 0.39	57.11 ± 0.66	48.00 ± 1.16
MWNet	$\mathbf{92.95} \pm \mathbf{0.33}$	86.46 ± 0.31	81.14 ± 0.94	$\underline{\mathbf{70.64} \pm \mathbf{0.31}}$	58.37 ± 0.33	$\underline{50.21 \pm 2.98}$
Soft-label	92.63 ± 0.27	$\underline{86.52 \pm 0.10}$	80.94 ± 0.25	70.50 ± 0.44	57.48 ± 0.43	48.18 ± 0.89
Gen-label	92.56 ± 0.56	84.68 ± 0.57	78.32 ± 0.94	70.46 ± 0.37	57.86 ± 0.50	48.08 ± 0.98
GDW	$\underline{\mathbf{92.94} \pm \mathbf{0.15}}$	$\mathbf{88.14} \pm \mathbf{0.35}$	$\mathbf{84.11} \pm \mathbf{0.21}$	$\mathbf{70.65} \pm \mathbf{0.52}$	$\mathbf{59.82} \pm \mathbf{1.62}$	$\mathbf{53.33} \pm \mathbf{3.70}$

Section 4.1 and Section 4.2, respectively. Furthermore, we conduct experiments on the real-world dataset Clothing1M [4] in Section 4.3.

4.1 Label Noise Setting

Setup. Following [23], we study two settings of label noise: a) Uniform noise: every instance's label uniformly flips to other class labels with probability p; b) Flip noise: each class randomly flips to another class with probability p. Note that the probability p represents the noise ratio. We randomly select 100 clean images per class from CIFAR10 [44] as the meta set (1000 images in total). Similarly, we select a total of 1000 images from CIFAR100 as its meta set. We use ResNet-32 [45] as the classifier model.

Comparison methods. We mainly compare GDW with meta-learning methods: 1) L2RW [22]: assign weights to instances based on gradient directions; 2) INSW [24]: derive instance weights adaptively from the meta set; 3) MWNet [23]; 4) Soft-label [39]: learn a label smoothing parameter for every instance; 5) Gen-label [37]: generate a meta-soft-label for every instance. We also compare some traditional methods: 6) BaseModel: train ResNet-32 on the noisy training set; 7) Fine-tuning: use the meta set to fine-tune the trained model in BaseModel; 8) Co-teach [20]; 9) GLC [19].

Training. Most of our training settings follow [23] and we use the cosine learning rate decay schedule [46] for a total of 80 epochs for all methods. See Appendix B for details.

Analysis. For all experiments, we report the mean and standard deviation over 5 runs in Table 2 and Table 3, where the best results are in **bold** and the second-best results are marked by underlines. First, we can observe that GDW outperforms nearly all the competing methods in all noise settings except for the 40% flip noise setting. Under this setting, GLC estimates the label corruption matrix well and thus performs the best, whereas the flip noise assumption scarcely holds in real-world scenarios. Note that GLC also performs much better than MWNet under the 40% flip noise setting as reported in [23]. Besides, under all noise settings, GDW has a consistent performance gain compared with MWNet, which aligns with our motivation in Figure 1. Furthermore, as the ratio increases from 40% to 60% in the uniform noise setting, the gap between GDW and MWNet increases from 1.68% to 2.97% in CIFAR10 and 1.45% to 3.12% in CIFAR100. Even under 60% uniform noise, GDW still

Table 3: Test accuracy on CIFAR10 and CIFAR100 with different flip noise ratios.

Dataset	CIFAR10			CIFAR100		
	0%	20%	40%	0%	20%	40%
BaseModel	92.73 ± 0.37	90.14 ± 0.35	81.20 ± 0.93	70.42 ± 0.54	64.96 ± 0.16	49.83 ± 0.82
Fine-tuning	92.77 ± 0.37	90.15 ± 0.36	81.53 ± 0.96	70.52 ± 0.57	65.02 ± 0.22	50.23 ± 0.71
Co-teach	91.54 ± 0.39	89.27 ± 0.24	69.77 ± 3.97	68.33 ± 0.13	62.96 ± 0.73	42.54 ± 1.68
GLC	90.85 ± 0.22	90.22 ± 0.13	**89.74 ± 0.19**	65.05 ± 0.59	64.11 ± 0.40	**63.11 ± 0.93**
L2RW	89.70 ± 0.50	88.21 ± 0.49	82.90 ± 1.27	63.40 ± 1.31	55.27 ± 2.27	45.41 ± 2.53
INSW	92.70 ± 0.57	89.90 ± 0.45	80.09 ± 2.00	70.52 ± 0.39	65.32 ± 0.27	50.13 ± 0.39
MWNet	**92.95 ± 0.33**	89.93 ± 0.17	85.55 ± 0.82	**70.64 ± 0.31**	64.72 ± 0.68	50.62 ± 0.46
Soft-label	92.63 ± 0.27	90.17 ± 0.47	85.52 ± 0.78	70.50 ± 0.44	65.20 ± 0.45	50.97 ± 0.41
Gen-label	92.56 ± 0.56	90.18 ± 0.13	80.93 ± 1.29	70.46 ± 0.37	64.94 ± 0.53	49.93 ± 0.55
GDW	**92.94 ± 0.15**	**91.05 ± 0.26**	87.70 ± 0.37	**70.65 ± 0.52**	**65.41 ± 0.75**	52.44 ± 0.79

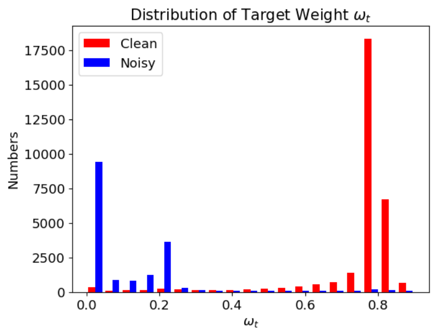
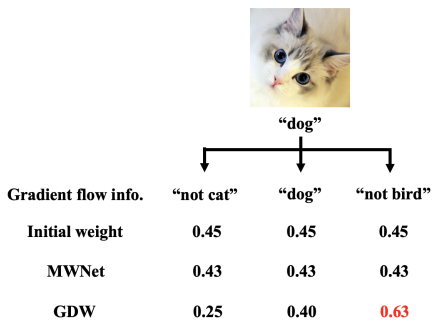


Figure 3: Class-level target weight (ω_t) distribution on CIFAR10 under 40% uniform noise. ω_t of most clean instances are larger than that of most noisy instances, which means ω_t can differentiate between clean and noisy instances.



Gradient flow info.	"not cat"	"dog"	"not bird"
Initial weight	0.45	0.45	0.45
MWNet	0.43	0.43	0.43
GDW	0.25	0.40	0.63

Figure 4: The change of class-level weights in an iteration for a noisy instance (cat mislabeled as "dog"). MWNet downweights all gradient flows. In contrast, GDW upweights the "not bird" gradient flow for better information use.

has low test errors in both datasets and achieves more than 3% gain in CIFAR10 and 6% gain in CIFAR100 compared with the second-best method. Last but not least, GDW outperforms Soft-label and Gen-label in all settings. One possible reason is that manipulating gradient flows is a more direct way to capture class-level information than learning labels.

In Figure 3, we show the distribution of class-level target weight (ω_t) on clean and noisy instances in one epoch. We observe that ω_t of most clean instances are larger than that of most noisy instances, which indicates that ω_t can distinguish between clean instances and noisy instances. This is consistent with Eq. 3 that ω_t serves as the instance weight.

To better understand the changing trend of non-target class-level weights, we visualize the ratio of increased weights in one epoch in Figure 5. Specifically, there are three categories: **n**on-**t**arget **w**eights on **c**lean instances (w^c_{nt}), **t**rue **t**arget **w**eights on **n**oisy instances (w^n_{tt}) and **n**on-**t**arget (excluding true targets) **w**eights on **n**oisy instances (w^n_{nt}). Note that in Figure 1, ω^n_{tt} represents the importance of the "not cat" gradient flow and ω^n_{nt} represents the importance of the "not bird" gradient flow. If the cat image in Figure 1 is correctly labeled as "cat", then the two non-target weights ω^c_{nt} are used to represent the importance of the "not dog" and the "not bird" gradient flows, respectively. In one epoch, we calculate **the ratios of** the number of increased w^c_{nt}, w^n_{tt} and w^n_{nt} **to** the number of all corresponding weights. w^c_{nt} and w^n_{nt} are expected to increase since their gradient flows contain valuable information, whereas w^n_{tt} is expected to decrease because the "not cat" gradient flow contains harmful information. Figure 5 aligns perfectly with our expectation. Note that the lines of w^c_{nt} and w^n_{nt} nearly coincide with each other and fluctuate around 65%. This means non-target weights on clean instances and noisy instances share the same changing pattern, i.e., around 65% of w^c_{nt} and w^n_{nt} increase. Besides, less than 20% of w^n_{tt} increase and thus more than 80% decrease, which means the gradient flows of w^n_{tt} contain much harmful information.

In Figure 4, we show the change of class-level weights in an iteration for a noisy instance, i.e., a cat image mislabeled as "dog". The gradient flows of "not cat" and "dog" contain harmful information and thus are downweighted by GDW. In addition, GDW upweights the valuable "not bird" gradient

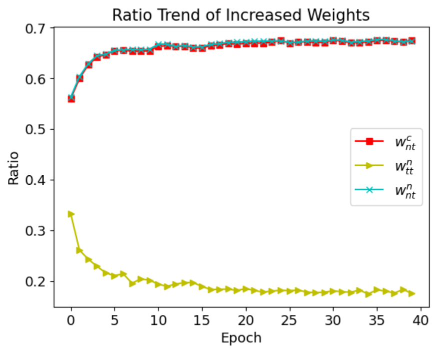


Figure 5: Ratio trend of the number of increased w^c_{nt}, w^n_{tt}, and w^n_{nt}. Around 65% of w^c_{nt} and w^n_{nt} increase since they contain useful information. Besides, less than 20% of w^n_{tt} increase and thus more than 80% of w^n_{tt} decrease since they contain harmful information.

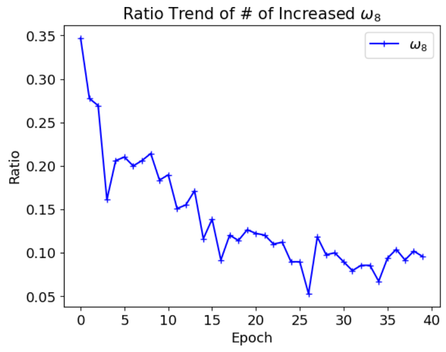


Figure 6: Ratio trend of the number of increased ω_8 on $C9$ instances. Less than 10% of ω_8 increase and thus more than 90% decrease. A small ω_8 strikes a balance between two kinds of information: "$C8$" and "not $C8$", which better handles class imbalance.

Table 4: Test accuracy on the long-tailed CIFAR10 and CIFAR100 with different imbalance ratios.

Dataset	CIFAR10			CIFAR100		
	$\mu = 1$	$\mu = 0.1$	$\mu = 0.01$	$\mu = 1$	$\mu = 0.1$	$\mu = 0.01$
BaseModel	92.73 ± 0.37	85.93 ± 0.57	69.77 ± 1.13	70.42 ± 0.54	56.25 ± 0.49	37.79 ± 0.82
Fine-tuning	92.77 ± 0.37	82.60 ± 0.49	59.76 ± 1.00	70.52 ± 0.57	55.95 ± 0.50	37.10 ± 0.87
Focal	91.68 ± 0.49	84.57 ± 0.83	65.78 ± 4.02	68.48 ± 0.38	55.02 ± 0.51	37.43 ± 1.00
Balanced	92.80 ± 0.47	86.05 ± 0.46	63.63 ± 3.60	70.56 ± 0.56	55.02 ± 0.80	27.60 ± 1.39
L2RW	89.70 ± 0.50	79.11 ± 3.40	51.15 ± 7.13	63.40 ± 1.31	46.28 ± 4.51	25.86 ± 5.78
INSW	92.70 ± 0.57	86.31 ± 0.28	70.27 ± 0.24	70.52 ± 0.39	55.94 ± 0.51	37.67 ± 0.59
MWNet	$\mathbf{92.95 \pm 0.33}$	86.17 ± 0.75	62.70 ± 1.76	$\mathbf{70.64 \pm 0.31}$	56.49 ± 1.52	37.83 ± 0.86
GDW	$\mathbf{92.94 \pm 0.15}$	$\mathbf{86.77 \pm 0.55}$	$\mathbf{71.31 \pm 1.03}$	$\mathbf{70.65 \pm 0.52}$	$\mathbf{56.78 \pm 0.52}$	$\mathbf{37.94 \pm 1.58}$

flow from 0.45 to 0.63. By contrast, unable to capture class-level information, MWNet downweights all gradient flows from 0.45 to 0.43, which leads to information loss on the "not bird" gradient flow.

Training without the zero-mean constraint. We have also tried training without the zero-mean constraint in Section 3.3 and got poor results. Denote the **t**rue **t**arget as tt and one of the **n**on-**t**arget labels as nt ($nt \neq tt$). Note that the gradient can be unrolled as (see Appendix C for details):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \omega_t \sum_j \left(p'_j - \mathbf{y}_j\right) \frac{\partial l_j}{\partial \boldsymbol{\theta}} + \left(\sum_k \omega_k p_k - \omega_t\right) \sum_j p'_j \frac{\partial l_j}{\partial \boldsymbol{\theta}}. \tag{13}$$

If $\sum_k \omega_k p_k - \omega_t$ is positive and the learning rate is small enough, $\left(\sum_k \omega_k p_k - \omega_t\right) p'_{tt} \frac{\partial l_{tt}}{\partial \boldsymbol{\theta}}$ contributes to the decrease of the true target logit l_{tt} after a gradient descent step. If negative, $\left(\sum_k \omega_k p_k - \omega_t\right) p'_{nt} \frac{\partial l_{nt}}{\partial \boldsymbol{\theta}}$ contributes to the increase of the non-target logit l_{nt}. Therefore, without the zero-mean constraint, the second term in Eq. 13 may hurt the performance of the model regardless of the sign of $\sum_k \omega_k p_k - \omega_t$. Similarly, training without the constraint results in poor performance in other settings. Hence we omit those results in the following subsections.

4.2 Class Imbalance Setting

Setup and comparison methods. The imbalance factor $\mu \in (0, 1)$ of a dataset is defined as the number of instances in the largest class divided by that of the smallest [23]. Long-Tailed CIFAR [44] are created by reducing the number of training instances per class according to an exponential function $n = n_i \mu^{i/(C-1)}$, where i is the class index (0-indexed) and n_i is the original number of training instances. Comparison methods include: 1) L2RW [22]; 2) INSW [24]; 3) MWNet [23]; 4) BaseModel; 5) Fine-tuning; 6) Balanced [13]; 7) Focal [21].

Table 5: Test accuracy on Clothing1M

Method	BaseModel	Fine-tuning	Co-teach	GLC	L2RW	INSW	MWNet	Soft-label	Gen-label	**GDW**
Accuracy(%)	65.02	67.68	68.13	68.60	<u>68.80</u>	68.25	68.46	68.69	67.64	**69.39**

Analysis. As shown in Table 4, GDW performs best in nearly all settings and exceeds MWNet by 8.6% when the imbalance ratio μ is 0.01 in CIFAR10. Besides, INSW achieves competitive performance at the cost of introducing a huge amount of learnable parameters (equal to the training dataset size N). Furthermore, we find that BaseModel achieves competitive performance, but fine-tuning on the meta set hurts the model's performance. We have tried different learning rates from 10^{-7} to 10^{-1} for fine-tuning, but the results are similar. One explanation is that the balanced meta set worsens the model learned from the imbalanced training set. These results align with the experimental results in [24] which also deals with class imbalance.

Denote the smallest class as $C9$ and the second smallest class as $C8$ in Long-Tailed CIFAR10 with $\mu = 0.1$. Recall ω_j denotes the j^{th} class-level weight. For all $C9$ instances in an epoch, we calculate **the ratio of** the number of increased ω_8 **to** the number of all ω_8, and then visualize the ratio trend in Figure 6. Since $C9$ is the smallest class, instance weighting methods upweight both ω_8 and ω_9 on a $C9$ instance. Yet in Figure 6, less than 10% of ω_8 increase and thus more than 90% decrease. This can be explained as follows. There are two kinds of information in the long-tailed dataset regarded to $C8$: "$C8$" and "not $C8$". Since $C8$ belongs to the minority class, the dataset is biased towards the "not $C8$" information. Because ω_8 represents the importance of "not $C8$", a smaller ω_8 weakens the "not $C8$" information. As a result, decreased ω_8 achieves a balance between two kinds of information: "$C8$" and "not $C8$", thus better handling class imbalance at the class level.

4.3 Real-world Setting

Setup and training. The Clothing1M dataset contains one million images from fourteen classes collected from the web [4]. Labels are constructed from surrounding texts of images and thus contain some errors. We use the ResNet-18 model pre-trained on ImageNet [47] as the classifier. The comparison methods are the same as those in the label noise setting since the main issue of Clothing1M is label noise [4]. All methods are trained for 5 epochs via SGD with a 0.9 momentum, a 10^{-3} initial learning rate, a 10^{-3} weight decay, and a 128 batchsize. See Appendix D for details.

Analysis. As shown in Table 5, GDW achieves the best performance among all the comparison methods and outperforms MWNet by 0.93%. In contrast to unsatisfying results in previous settings, L2RW performs quite well in this setting. One possible explanation is that, compared with INSW and MWNet which update weights iteratively, L2RW obtains instance weights only based on current gradients. As a result, L2RW can more quickly adapt to the model's state, but meanwhile suffers from unstable weights [23]. In previous settings, we train models from scratch, which need stable weights to stablize training. Therefore, INSW and MWNet generally achieve better performance than L2RW. Whereas in this setting, we use the pre-trained ResNet-18 model which is already stable enough. Thus L2RW performs better than INSW and MWNet.

5 Conclusion

Many instance weighting methods have recently been proposed to tackle label noise and class imbalance, but they cannot capture class-level information. For better information use when handling the two issues, we propose GDW to generalize data weighting from instance level to class level by reweighting gradient flows. Besides, to efficiently obtain class-level weights, we design a two-stage weight generation scheme embedded in the three-step bi-level optimization framework. To be specific, this scheme leverages intermediate gradients to update class-level weights via a gradient descent step. In this way, GDW achieves remarkable performance improvement in various settings.

References

[1] Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. <u>arXiv preprint arXiv:2007.08199</u>, 2020.

[2] Haibo He and Edwardo A Garcia. Learning from imbalanced data. IEEE Transactions on knowledge and data engineering, 21(9):1263–1284, 2009.

[3] Nancy E. ElHady and Julien Provost. A Systematic Survey on Sensor Failure Detection and Fault-Tolerance in Ambient Assisted Living. Sensors, 18(7), July 2018. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[4] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. Learning from massive noisy labeled data for image classification. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2691–2699, 2015.

[5] Görkem Algan and İlkay Ulusoy. Label Noise Types and Their Effects on Deep Learning. arXiv:2003.10471 [cs], March 2020. arXiv: 2003.10471.

[6] Xingquan Zhu and Xindong Wu. Class Noise vs. Attribute Noise: A Quantitative Study. Artif. Intell. Rev., 22:177–210, November 2004.

[7] Benoit Frenay and Michel Verleysen. Classification in the Presence of Label Noise: A Survey. IEEE Transactions on Neural Networks and Learning Systems, 25(5):845–869, May 2014. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

[8] Qiuye Zhao and Mitch Marcus. Long-Tail Distributions and Unsupervised Learning of Morphology. In Proceedings of COLING 2012, pages 3121–3136, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.

[9] Grant Van Horn and Pietro Perona. The Devil is in the Tails: Fine-grained Classification in the Wild. arXiv:1709.01450 [cs], September 2017. arXiv: 1709.01450.

[10] Harshita Patel, Dharmendra Singh Rajput, G Thippa Reddy, Celestine Iwendi, Ali Kashif Bashir, and Ohyun Jo. A review on classification of imbalanced data for wireless sensor networks. International Journal of Distributed Sensor Networks, 16(4):1550147720916404, April 2020. Publisher: SAGE Publications.

[11] Reyes Pavón, Rosalía Laza, Miguel Reboiro-Jato, and Florentino Fdez-Riverola. Assessing the Impact of Class-Imbalanced Data for Classifying Relevant/Irrelevant Medline Documents. In Miguel P. Rocha, Juan M. Corchado Rodríguez, Florentino Fdez-Riverola, and Alfonso Valencia, editors, 5th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2011), Advances in Intelligent and Soft Computing, pages 345–353, Berlin, Heidelberg, 2011. Springer.

[12] Qi Dong, Shaogang Gong, and Xiatian Zhu. Class Rectification Hard Mining for Imbalanced Deep Learning. arXiv:1712.03162 [cs], December 2017. arXiv: 1712.03162.

[13] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples, 2019.

[14] Saptarshi Sinha, Hiroki Ohashi, and Katsuyuki Nakamura. Class-Wise Difficulty-Balanced Loss for Solving Class-Imbalance. In Hiroshi Ishikawa, Cheng-Lin Liu, Tomas Pajdla, and Jianbo Shi, editors, Computer Vision – ACCV 2020, volume 12627, pages 549–565. Springer International Publishing, Cham, 2021. Series Title: Lecture Notes in Computer Science.

[15] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. Journal of Big Data, 6(1):1–54, 2019.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2818–2826, Las Vegas, NV, USA, June 2016. IEEE.

[17] J. Goldberger and E. Ben-Reuven. Training deep neural-networks using a noise adaptation layer. In ICLR, 2017.

[18] Tongliang Liu and Dacheng Tao. Classification with Noisy Labels by Importance Reweighting. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(3):447–461, March 2016. arXiv: 1411.7718.

[19] Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. Using Trusted Data to Train Deep Networks on Labels Corrupted by Severe Noise. arXiv:1802.05300 [cs], January 2019. arXiv: 1802.05300.

[20] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust Training of Deep Neural Networks with Extremely Noisy Labels. arXiv:1804.06872 [cs, stat], October 2018. arXiv: 1804.06872.

[21] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal Loss for Dense Object Detection. arXiv:1708.02002 [cs], February 2018. arXiv: 1708.02002.

[22] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to Reweight Examples for Robust Deep Learning. arXiv:1803.09050 [cs, stat], May 2019. arXiv: 1803.09050.

[23] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting. arXiv:1902.07379 [cs, stat], September 2019. arXiv: 1902.07379.

[24] Zhiting Hu, Bowen Tan, Ruslan Salakhutdinov, Tom Mitchell, and Eric P. Xing. Learning Data Manipulation for Augmentation and Weighting. arXiv:1910.12795 [cs, stat], October 2019. arXiv: 1910.12795.

[25] Xinyi Wang, Hieu Pham, Paul Michel, Antonios Anastasopoulos, Jaime Carbonell, and Graham Neubig. Optimizing Data Usage via Differentiable Rewards. arXiv:1911.10088 [cs, stat], June 2020. arXiv: 1911.10088.

[26] Arash Vahdat. Toward Robustness against Label Noise in Training Deep Discriminative Neural Networks. arXiv:1706.00038 [cs, stat], November 2017. arXiv: 1706.00038.

[27] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16:321–357, June 2002. arXiv: 1106.1813.

[28] Salman H. Khan, Munawar Hayat, Mohammed Bennamoun, Ferdous Sohel, and Roberto Togneri. Cost Sensitive Learning of Deep Feature Representations from Imbalanced Data. arXiv:1508.03422 [cs], March 2017. arXiv: 1508.03422.

[29] Charles Elkan. The foundations of cost-sensitive learning. In In Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pages 973–978, 2001.

[30] Ashish Anand, Ganesan Pugalenthi, Gary B. Fogel, and P. N. Suganthan. An approach for classification of highly imbalanced data using weighting and undersampling. Amino Acids, 39(5):1385–1391, November 2010.

[31] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. arXiv:1806.04910 [cs, stat], July 2018. arXiv: 1806.04910.

[32] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. Science, 350(6266):1332–1338, December 2015. Publisher: American Association for the Advancement of Science Section: Research Article.

[33] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. arXiv:1806.09055 [cs, stat], April 2019. arXiv: 1806.09055.

[34] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Learning to model the tail. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.

[35] Junnan Li, Yongkang Wong, Qi Zhao, and Mohan S. Kankanhalli. Learning to Learn From Noisy Labeled Data. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 5046–5054, Long Beach, CA, USA, June 2019. IEEE.

[36] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. MentorNet: Learning Data-Driven Curriculum for Very Deep Neural Networks on Corrupted Labels. arXiv:1712.05055 [cs], August 2018. arXiv: 1712.05055.

[37] Görkem Algan and Ilkay Ulusoy. Meta Soft Label Generation for Noisy Labels. arXiv:2007.05836 [cs, stat], January 2021. arXiv: 2007.05836.

[38] Zizhao Zhang, Han Zhang, Sercan O. Arik, Honglak Lee, and Tomas Pfister. Distilling Effective Supervision from Severe Label Noise. arXiv:1910.00701 [cs, stat], June 2020. arXiv: 1910.00701.

[39] Nidhi Vyas, Shreyas Saxena, and Thomas Voice. Learning Soft Labels via Meta Learning. arXiv:2009.09496 [cs, stat], September 2020. arXiv: 2009.09496.

[40] Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. Data Parameters: A New Family of Parameters for Learning a Differentiable Curriculum. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 11095–11105. Curran Associates, Inc., 2019.

[41] Zhenyue Qin, Dongwoo Kim, and Tom Gedeon. Rethinking Softmax with Cross-Entropy: Neural Network Classifier as Mutual Information Estimator. arXiv:1911.10688 [cs, stat], September 2020. arXiv: 1911.10688.

[42] Shuai Zhao, Liguang Zhou, Wenxiao Wang, Deng Cai, Tin Lun Lam, and Yangsheng Xu. Towards Better Accuracy-efficiency Trade-offs: Divide and Co-training. arXiv:2011.14660 [cs], March 2021. arXiv: 2011.14660 version: 3.

[43] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv:1704.04861 [cs], April 2017. arXiv: 1704.04861 version: 1.

[44] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. University of Toronto, 2009.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[46] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983, 2016.

[47] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] The proposed method can only be applied on classification tasks.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 3.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section 3.

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We only use public datasets and the code is in the supplementary materials.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Most of our settings follow [23] and other details are in the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We repeat all experiments on CIFAR10 and CIFAR100 with five different seeds and the mean and standard deviation are reported. For the Clothing1M dataset, we only run one experiment due to limited resources.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We use one V100 GPU. See the appendix for details.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] For dataset, we cite the papers of CIFAR datasets and the Clothing1M dataset. For code, we cite [45].
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]