

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/382557715>

Reimplementation of Learning to Reweight Examples for Robust Deep Learning

Preprint · July 2024

DOI: 10.13140/RG.2.2.33409.67680

CITATIONS

0

READS

42

5 authors, including:



[Deerajkumar Parthipan](#)

Purdue University West Lafayette

5 PUBLICATIONS 3 CITATIONS

SEE PROFILE

Reimplementation of Learning to Reweight Examples for Robust Deep Learning

Deerajkumar Parthipan* Parth Patil* Ben Boardley* Jack Gardner* Emily Loiselle*

Abstract

Deep neural networks (DNNs) have been used to create models for many complex analysis problems like image recognition and medical diagnosis. DNNs are a popular tool within machine learning due to their ability to model complex patterns and distributions. However, the performance of these networks is highly dependent on the quality of the data used to train the models. Two characteristics of these sets, noisy labels and training set biases, are known to frequently cause poor generalization performance as a result of overfitting to the training set. This paper aims to solve this problem using the approach proposed by Ren et al. (2018) using meta-training and on-line weight approximation. We will first implement a toy-problem to crudely verify the claims made by the authors of Ren et al. (2018) and then venture into using the approach to solve a real world problem of Skin-cancer detection using imbalanced image dataset.

1. Problem Statement

Label noise in the training of these models is caused mainly by the lack of high-quality labels which can only be produced by extensive human inspection. This misclassification of data can be done either through attacks or unintentional classification. Deep neural networks can be prone to adversarial attacks intended to make the model respond inaccurately, typically in the form of dataset poisoning which is done by inputting carefully crafted data to make the model behave in unintended ways. Additionally, Frenay’s research summarizes the main sources for unintentional noise: insufficient information, subjective classification, and general mistakes (Frénay & Verleysen, 2014). As for training set biases, they are often caused by a disproportionate representation of certain classes in the labeled dataset which lead to the model weighting these classes differently. This can be formulated as the training set distribution $P(x, y)$ not aligning with the real world or evaluation distribution $P(x^k, y^k)$ (Ren et al., 2018). Training set biases can also take place when the nature of the data is imbalanced, an example of this is

in autonomous driving, where, although very important in classification, emergency vehicles and animals are not seen as frequently in the dataset, and thus the model may not be well fine-tuned to these classes.

Current solutions to these issues involve small and costly adjustments of hyperparameters to increase or decrease the importance of certain types of labels. Reweighting of training examples has its roots in classical statistical algorithms used to downplay the loss generated by outliers. This methodology of reweighting is also useful in various areas of machine learning like LSTMs and curriculum learning. These methods often have contradictory solutions because they are loss based. These loss based methods optimize hyperparameters by solving the equation below

$$\lambda^* = \arg \min_{\lambda} f(\lambda, D_{\text{train}}, D_{\text{test}}) \quad (1)$$

where the function $f()$ measures the loss of a model generated by a specific algorithm with hyperparameters λ on training data D_{train} and evaluated on validation data D_{test} .

This hyper-parameter tuning is not only expensive but it can have contradictions when correcting for noise versus training set biases. When trying to correct for label noise many of these algorithms put a larger weight on examples that have smaller training losses as those are typically the cleaner images. However, when accounting for imbalance in datasets, common solutions will aim to give more weight to examples with larger training loss as those are commonly the underrepresented class, which can be seen in many boosting algorithms (e.g. AdaBoost). Due to the contradiction of the solutions, there are large challenges posed by datasets that are both noisy and imbalanced.

The effects of noise on an imbalanced data set is explored in an experiment in which Medicare Part B fraud detection data is injected with label noise (Kennedy et al., 2021). This experiment showed that amongst different machine learning algorithms (Linear Regression, Multi-layer Perception, Random Forest and XGBoost) all of them saw

significant decreases in performance with injected label noise, including a 20% drop in Area Under Precision Recall Curve (AUPRC) by the linear regression and an 8% drop in both XGBoost and Random Forest. The paper being reviewed aims to help provide an efficient solution for datasets of this nature.

The proposed solution will aim to provide a more robust approach to solving the problems faced when using lower-quality datasets under the assumption that there is a small high quality validation set that is representative of the task. This validation set is used to guide the example weighting through each training iteration by finding the optimal example weights that minimize the validation loss (Ren et al., 2018). This solution helps solve a research gap as many current solutions to this issue require extensive learned hyper-parameters (1) which can be expensive during training as fine-tuning hyper-parameters is an offline procedure. The proposed solution will help create an online training approach without any additional hyper-parameters.

2. Method

The proposed solution aims to provide a more robust approach to solving the problems faced when using biased or noisy labeled dataset, under the assumption that there is a small high quality validation set that is representative of the task. This validation set is used to guide the example weighting through each training iteration by finding the optimal example weights that lead to model parameters that minimize the validation loss (Ren et al., 2018).

More formally, the model fits a meta-learning objective that can be implemented along with traditional supervised learning. Assume that there are both a training set $\{(x_i, y_i, 1 \leq i \leq N)\}$ and a validation set $\{(x_i^v, y_i^v, 1 \leq i \leq M)\}$ in which $M \ll N$. (Ren et al., 2018). The objective of this method is to minimize the weighted expected loss of the model in which $\theta^*(w)$ is the optimal model parameters given the weight vector \hat{w} and a loss function $J(\theta)$.

$$\theta^*(w) = \arg \min_{\theta} \sum_{i=1}^M w_i J_i(\theta), \quad (2)$$

The weighted vector \hat{w} is another optimized parameter. The objective is to find optimal example weights that will lead to the θ that minimizes the loss on the validation set. In which J_i^v is the loss with respect to the validation set and $\theta^*(w)$ is with respect to equation 1.

$$w^* = \arg \min_{w, w \geq 0} \frac{1}{M} \sum_{i=1}^M J_i^v(\theta^*(w)). \quad (3)$$

However, as currently formulated this meta learning objective can be interpreted as nested optimization loops as shown in equation 3, which can be quite expensive as it requires iterations over the dataset for every single loop.

$$w^* = \arg \min_{w, w \geq 0} \frac{1}{M} \sum_{i=1}^M J_i^v(\arg \min_{\theta} \sum_{i=1}^N w_i J_i(\theta)). \quad (4)$$

In order to solve this inefficiency, the paper adopts an online approach to calculating the weights using gradient descent. Following a similar approach to traditional Stochastic Gradient Descent (SGD), this training method involves a descent step for both θ and the weights, which will be represented by ϵ . Allow $J_{\epsilon, n}(\theta) = \epsilon_n * J_n(\theta)$ and allow B to be the current training batch at timestep t . Thus allowing us to manipulate traditional SGD to the below equation.

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{1}{B} \sum_{n=1}^B \nabla_{\theta} J_{\epsilon, n}(\theta) \quad (5)$$

The optimal weight for example n at time step t can then be estimated by a gradient descent step on ϵ_n at the same time step with respect the to loss on the validation set of size M .

$$x_{n, t} = -\eta * \frac{\partial}{\partial \epsilon_{n, t}} \frac{1}{M} \sum_{i=1}^M J_i^v(\theta_{t+1}(\epsilon)) \quad (6)$$

However negative weights can create instability during optimization so w is estimated by.

$$\tilde{w}_{n, t} = \max(x_{n, t}, 0) \quad (7)$$

Due to the fact that example weights need to add to 1, the weight can be normalized to provide this functionality. In which δ is the Dirac delta function to prevent an undefined estimate.

$$w_{n, t} = \frac{\tilde{w}_{n, t}}{\|\tilde{w}_t\|_1 + \delta(\|\tilde{w}_t\|_1)} \quad (8)$$

Using the gradient descent step on $J^v(\theta)$ with respect to the weights it is possible to reweight examples in a single iteration allowing us to convert the previous meta-learning objective into an online learning procedure. Helping create a robust training implementation without the need to tune hyperparameters offline.

2.1. Related Work

This solution helps solve a research gap as many current meta-learning solutions to this issue, such as MAML

(Finn et al., 2017) and few-shot learning (Ren et al., 2018), require learned hyper-parameters which can be expensive during training and thus must be trained offline. The proposed solution will help create an online training approach without any additional hyper-parameters.

Another issue among current algorithms that are used to address these distorted datasets is the contradicting approaches for problems with training sets that are noisy and sets that have imbalanced classes. To solve noisy label training sets, classical algorithms tend to favor examples with less training loss because they are more likely to be clean, but solutions to fix class imbalance in datasets, such as hard negative mining (Malisiewicz, 2011), give more weight to examples with higher training loss because they are usually the underrepresented class. By requiring a small unbiased validation set to guide training, so as not to rely on loss calculated completely with the biased training set to set weights, this method allows the proposed solution to resist overfitting noise better compared to previous literature.

2.2. Learning to reweight examples in a MLP (Multi-Layer Perceptron Network)

This example gives an algorithm to compute $w_{n,t}$ within a MLP network, for which we calculate the gradients of the validation loss w.r.t the local perturbation (ϵ). The MLP is defined for parameters for each layer, $\theta = (\theta_l)_{l=1}^L$, where L is the number of layers. For each layer, z^l is the pre-activation value computed as a weighted sum of inputs, then a non-linear activation function σ is applied to obtain the post-activation \tilde{z}_l

$$\begin{aligned} z_l &= \theta^T \tilde{z}_{l-1} \\ \tilde{z}_l &= \sigma(z_l) \end{aligned}$$

Now, the algorithm computes the gradients of the validation loss with respect to the perturbation (ϵ). This calculation involves how adjusting the weights of examples can optimise the training process. These gradients of the validation loss are approximated by local dot products of the gradients of loss w.r.t. z_l (g_l) and the gradients w.r.t. θ_l ($\tilde{z}_{l-1} g_l^T$).

$$\begin{aligned} & \frac{\partial}{\partial \epsilon_{n,t}} \mathbb{E}[J^v(\theta_{t+1}(\epsilon)) | \epsilon_{i,t}=0] \\ &= -\frac{1}{m} \sum \sum (z^{v_{j,l-1}T} \tilde{z}_{i,l-1}) (g^{v_{j,l}T} g_{i,l}) \end{aligned}$$

This equation shows that the gradient is composed on two terms, the difference in pre-activation values between validation and training, and the difference in gradient directions (g_l) for validation and training. This helps us to conclude that, if a pair of training and validation examples are similar and provide similar gradient directions, then the training

example is considered to be beneficial and is up-weighted and conversely, if they provide opposite gradient directions, it is down weighted.

2.3. Implementation using Automatic Differentiation

The unnormalized weights of examples can be determined by summing up the correlations between the gradients. It leverages automatic differentiation to calculate the gradient of validation loss w.r.t. the example weights in the current batch. This method unrolls the gradient graph of the training batch and employs backward on backward differentiation to obtain second order gradients. The process followed by the learning to reweight examples using automatic differentiation algorithm is as follows:

- Perform a forward pass
- Calculate training loss
- Perform backward pass and compute gradients w.r.t. model parameters
- Calculate validation loss and gradients using automatic differentiation
- Update the example weights based in these gradients
- Update the model parameters using gradients from the training loss.

Because the automatic differentiation method calls for extra computational steps such as two full forward and backward passes on both the validation and training sets to calculate losses and gradients. Also the back on backward pass adds extra complexity. Due to this the total training period is about 3 times longer than with ordinary training, since backward passes also takes the same time as a forward pass.

2.4. Proof of Convergence of the Reweighted Training

The paper put forth two theorems to prove the convergence properties of the proposed method. The discussion on proofs can be found in the Appendix section of the paper.

1. Theorem 1: The proposed algorithm converges.
2. Theorem 2: The rate of convergence within ϵ is less than $O(1/\epsilon^2)$

3. Experiment

3.1. Toy Problem: Implementation Summary

Sections 3.1-3.4 aim to reimplement the reweighted model using a toy problem. The toy example we use to display the robust effects of this algorithm is an image classification task

using the CIFAR-10 dataset. We use a simple convolutional neural network (CNN) model with two different training schemes: the reweighting algorithm and the more traditional stochastic gradient descent (SGD) algorithm. Upon analysis, our replication of the proposed model was able to accurately reproduce the results and generate a more robust model than the traditional model.

3.2. Toy Problem: Dataset

The CIFAR-10 dataset includes ten classes, each with 6,000 images. This dataset is commonly used as a benchmark because of its low-resolution images which pose a challenge for models to learn features. Also its diverse set of classes make it suitable for testing the generalization capabilities of models. In our analysis, the distribution of these images were manipulated to simulate biased training sets.

3.3. Toy Problem: Implementation Details

The architecture of our simple CNN model consists of three convolutional layers followed by a max pooling layer to downsample the number of datapoints in the feature map to reduce computational complexity. This is then followed by two fully connected layers.

In our implementation we compare the performance of traditional SGD with that of the paper, learning to reweight model parameters. In this experiment we utilize an imbalanced CIFAR10 dataset. In the traditional SGD approach we use 60 epochs with a learning rate of 0.001 and momentum of 0.9. Similarly, the model trained with reweighting uses the same SGD learning rate and momentum. However, in each epoch the algorithm different from the traditional SGD, before we take descent steps on the model parameters we utilize meta-learning to optimize the weights of each sample's loss to minimize the error on the representative validation set. In order to implement meta-learning we leveraged the higher library which allows us to create a meta model that can be used to optimize the weight vector, without effecting the true model's parameters. After optimizing the weights, we then took a gradient descent step on the true model, using the weighted loss. Our code for the implementation can be found in the citations here (Team19, 2024).

3.4. Toy Problem: Results

We analyzed the effects of the reweighting algorithm by simulating different amounts of training set bias for a given class. In Figure 1, the reweighting trained model's accuracy is compared to the SGD trained model as the percentage of the training dataset for a single class is increased. For example, at the 60% mark, 60% of the training data is

associated with a single class while the remaining 40% is split evenly among the other nine classes. The model that used the reweighting training algorithm was able to outperform the control SGD trained model's accuracy in each test between 50-90%. It can also be seen that as the amount of class bias increases, the difference in accuracy increases even further.

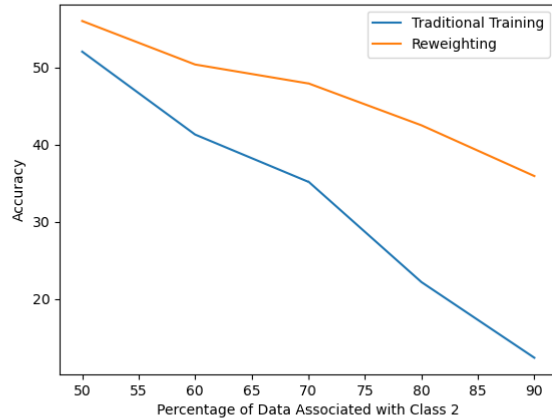


Figure 1. Accuracy of Model Trained with SGD vs Model Trained with Reweighting Algorithm

3.5. Toy Problem: Conclusion

The implementation demonstrates the ability of the reweighted method to produce better results on improperly scaled datasets compared to traditional methods. While the accuracy attained by the model in this paper was much lower than the one proposed by Ren et al. (2018), this discrepancy can be explained by the fact that our resources only allowed us to train and test on the CIFAR-10 dataset rather than larger equivalent datasets like CIFAR-1000. Additionally, the model in Ren et al. (2018) was only trained and tested on image recognition tasks with 2 classes rather than 10. Going forward, our next step would be to demonstrate how our model can perform better than traditional models on datasets with increased noise.

3.6. Real-world Problem Description

Sections 3.6-3.10 aims to reimplement the reweighted model using a real-world problem. Skin cancer can be deadly if not detected and treated early and accurately. Thus, an accurately trained model that automates diagnosis of suspicious lesions would be able to help patients seek medical intervention sooner, decreasing the risks for patients earlier on. Additionally, automatic diagnosis of skin lesions can help with consistency, healthcare resource allocation, and ease of access to quick diagnoses.

Oftentimes, human diagnosis can be subjective and prone to errors, leading to both overdiagnosis and underdiagnosis. Using automated diagnosis models will allow for more consistent and objective assessment. This automated system can then also be easily deployed widely, allowing patients to quickly obtain initial diagnoses without needing appointments. This in turn would be able to help hospitals optimize resource usage, as the need for extensive manual review of screenings and tests is decreased.

One issue faced oftentimes when creating an automated diagnosis model is class imbalance in the datasets available. This imbalance is caused because several types of lesions are much more rare compared to other types. In many of the modern algorithms used to train models, this imbalance may cause much higher inaccuracy for these minority classes. Additionally, these datasets can be prone to inaccurate labelling due to human error which can further hurt the accuracy of the model.

In order to deal with this class imbalance, we will use the training algorithm described by [Ren et al. \(2018\)](#) to dynamically adjust the loss function during training.

3.7. Real-world Dataset

The dataset employed for this task is the HAM10000 ("Human against Machine with 10000 training images") dataset. This dataset consists of 10015 dermatoscopic images that contain a characteristic set of 7 different diagnostic classes, collected from multiple populations. These classes include "Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec), basal cell carcinoma (bcc), benign keratosis-like lesions (solar lentigines / seborrheic keratoses and lichen-planus like keratoses, bkl), dermatofibroma (df), melanoma (mel), melanocytic nevi (nv) and vascular lesions (angiomas, angiokeratomas, pyogenic granulomas and hemorrhage, vasc)" ([Tschandl, 2018](#)). This dataset was chosen as our algorithm for reweighting examples should be well equipped to handle a dataset that has the imbalanced nature seen in this dataset. The training samples per class can be seen in Figure 2.

3.8. Real-world: Intended Outcome

Through this experimentation we would like to prove that learning to reweight examples can provide results beyond that of traditional Stochastic Gradient Descent (SGD) and other reweighting algorithms (e.g. boosting algorithms). In addition we hope that our re-implemented model can be competitive with the top 10 performances of task 3, lesion diagnosis, in the 2018 ISIC Competition, as the testing ground truth has been released. The balanced multi class accuracy in the top 10 range from 78% to 88% on the leader-

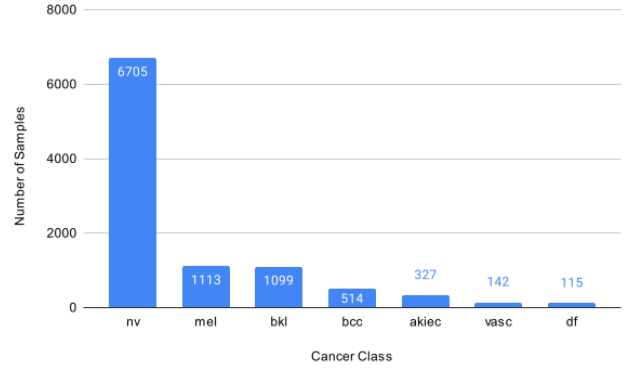


Figure 2. Distribution of training dataset

board ([Com](#)).

3.9. Real-world Problem: Experimental Setup

In our implementation of models built to solve this classification, we use three schemes for training: the reweighting algorithm described in ([Ren et al., 2018](#)), a conventional stochastic gradient descent (SGD) algorithm without any weighting, and an algorithm with a static weighted sampler implemented using Pytorch's weighted RandomSampler. For all of these schemes, our models were built by fine-tuning a pretrained Resnet50 model. The un-weighted and weighted sampler schemes were trained with 20 epochs while the model trained with the reweighting algorithm underwent 8 epochs. All 3 models had a learning rate of 0.001 and momentum of 0.9 and used stochastic gradient descent to optimize their parameters. In the weighted sampler model, the imported algorithm ensures that all data points are used during training but adjusts the probability of selecting each class in each of the training batches based on the distribution of classes in the training set. These weights differ from our reweighting implementation as it is an offline process with pre-calculated probabilities selected before training while our implementation adjusts the weights during the training process. The comparison between these three approaches will help demonstrate the effectiveness of the learning to reweight parameters algorithm.

The reweighting algorithm was implemented using a similar approach as was previously described in the toy problem in [subsection 3.3](#). Additionally, we extracted two different validation sets for training purposes. The first validation set was a small balanced subset used to train our example weighting, consisting of 3 samples per class. The second validation subset was used to test the generalization of our model after each epoch. This subset was also balanced in order to impartially generalize the overall accuracy of the model with 15 samples per class.

For testing, we utilized the 2018 ISIC challenge’s test data set and ground truth. We used a balanced multi-class accuracy function to match the metric used by the challenge’s official submission website. This involves averaging the individual accuracy of each class. Our Implementation can be found at the github repository cited in this paper (Gardner et al., 2024).

3.10. Real-world Problem: Results

The balanced multi-class accuracy of the different training methods can be seen in Table 1. The performance of our implemented algorithm exceeds the performance of weighted random sampler and the unweighted training by 48.87% and 51.49% respectively. The implemented method can be seen to be an effective approach for handling class imbalance. However, we were unable to produce results competitive with the top 10 in the leader-board of the ISIC Challenge in which the cutoff was 78%, however with more resources, training time, and experimentation with different models the performance may have been able to improve.

Table 1. Balanced Multi-class Accuracy

Model	Learning Method	Accuracy(%)
ResNet50	Learning to Reweight	65.56
ResNet50	Weighted Sampler	16.69
ResNet50	Unweighted Training	14.07

In Table 2 we can see that the learning to reweight algorithm has a better distribution of class accuracy than the other methodologies. The other two training techniques had several classes with a 0% accuracy, which is most likely attributed to the imbalanced nature of the problem. Our algorithms dynamic adjustment of weights during training seems to be an effective way of dealing with this difficulty. Additionally, our algorithms success can also be attributed to the small balanced validation set that the example weights are trained on.

In figure 3 and 4 we can see the confusion matrices of the unweighted learning algorithm and the learning to reweight algorithm. In these figures we can see the diagonal (correct predictions) of the reweight algorithm is more dominant than in the unweighted algorithm. We can also see that the unweighted algorithm commonly mistook the nv class for bkl and bcc classes while the reweighting algorithm seemed to correct this misconception.

Table 2. Summary of Testing Accuracy for Different Classes

Class	Learning to Reweight (%)	Unweighted Training (%)	Weighted Sampler (%)
nv	74.48	7.04	86.25
mel	69.59	0.0	0.0
bkl	54.38	39.63	26.27
bcc	80.65	49.46	4.30
akiec	55.81	2.33	0.0
vasc	62.86	0.0	0.0
df	61.3	0.0	0.0

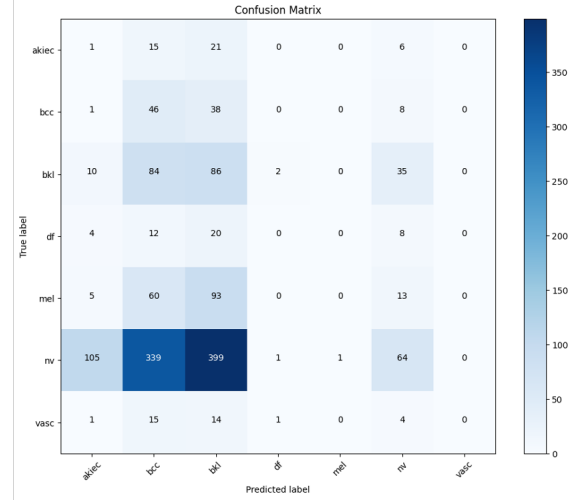


Figure 3. Confusion Matrix for Unweighted Training

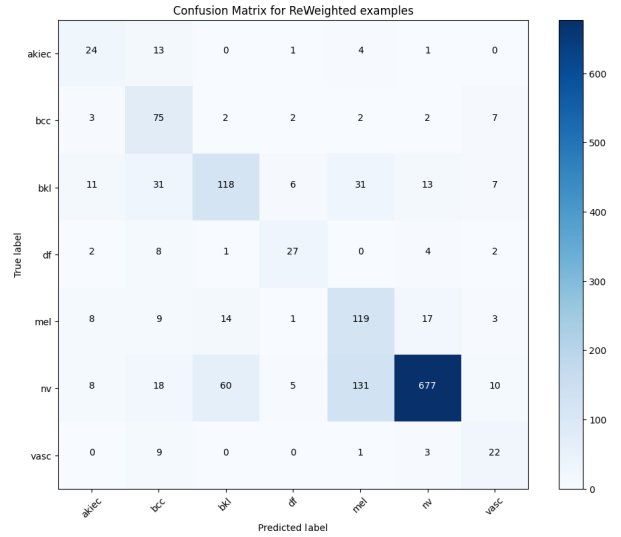


Figure 4. Confusion Matrix for Learning to Reweight Training

4. Conclusion

Our results show that the re-weighting algorithm we have re-implemented is able to effectively handle class imbalance. We have demonstrated that this approach significantly outperforms traditional training methods like SGD and Weighted Random Sampling. Potential progressions of our implementation would be to test the effectiveness of handling a dataset with a combination of both label noise and class imbalance and compare it more thoroughly with other possible solutions. The main feature of the real-world problem addressed in this paper is class imbalance, although label noise could have also been introduced into the dataset from incorrect diagnosis caused by human error. Finding a dataset with natural label noise or creating one through random poisoning could help us better see the effects of the reweighting algorithm.

Additionally, our algorithm, as mentioned previously, was unable to achieve competitive results compared to the top performers of the official 2018 ISIC Challenge. This displays the fact that even with a training algorithm that is able to do well against issues like class imbalance and label noise, there are still many other variables and approaches that need to be refined when creating a model. Still, with more resources and experimentation, we believe that our approach could achieve higher accuracy and improve its standing in the leader-board.

In conclusion, our study contributes to the growing body of literature on addressing class imbalance in machine learning applications, particularly in medical diagnosis. By re-implementing and validating the learning to reweight algorithm’s findings, we provide a valuable tool for researchers to improve the accuracy and reliability of automated diagnostic systems.

References

- ISIC Challenge 2018. <https://challenge.isic-archive.com/data/#2018>. Accessed: 2024-04-09.
- Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL <http://arxiv.org/abs/1703.03400>.
- Frénay, B. and Verleysen, M. Classification in the presence of label noise: A survey. *Neural Networks and Learning Systems, IEEE Transactions on*, 25:845–869, 05 2014. doi: 10.1109/TNNLS.2013.2292894.
- Gardner, J., Boardley, B., et al. Ece 50024 final project code. <https://github.com/jgardner21/MLProject>, 2024.
- Kennedy, R. K. L., Johnson, J. M., and Khoshgoftaar, T. M. The effects of class label noise on highly-imbalanced big data. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1427–1433, 2021. doi: 10.1109/ICTAI52525.2021.00227.
- Malisiewicz, T. Exemplar-based representations for object detection, association and beyond, 2011.
- Ren, M., Zeng, W., Yang, B., and Urtasun, R. Learning to reweight examples for robust deep learning. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4334–4343. PMLR, 10–15 Jul 2018. URL <https://proceedings.mlr.press/v80/ren18a.html>.
- Team19. Reweighting for Deep Learning. https://github.com/Parth1811/reweighting_for_deep_learning, 2024.
- Tschandl, P. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions, 2018. URL <https://doi.org/10.7910/DVN/DBW86T>.

5. Appendix

5.1. Theorem 1

Assuming the loss function is Lipschitz-smooth with constant L , have σ -bounded gradients and learning rate $\alpha_t = \frac{2n}{L\sigma^2}$, it can be shown that,

$$G(\theta_t + 1) \leq G(\theta_t) \quad (9)$$

$$\text{Where, } G(\theta_t) = \frac{1}{M} \sum_{i=1}^M J_i^v(\theta_{t+1}(\epsilon)) \quad (10)$$

Further, the theorem also states a strong relation for the expectation of the loss function. It says that for any time step, the expectation of loss function is equal to the value of the previous time step if and only if the gradient of loss is equal to zero; that is, convergence only happens at a minima. Mathematically $E[G(\theta_{t+1}) = G(\theta_t)] \Leftrightarrow \nabla G(\theta_t) = 0$

The proof for this theorem is two parts. First, the authors use the Lipschitz-smoothness of the loss function to establish this inequality

$$G(\theta_t + 1) \leq G(\theta_t) + \nabla G^T \Delta\theta + \frac{L}{2} \|\theta\|^2 \quad (11)$$

This equation, upon simplifications using the algorithm's $\Delta\theta$, yields the equation (12) below. Here τ_t is positive and so is $1 - \frac{L\alpha_t\sigma^2}{2n}$ because of the assumption regarding learning rate.

$$G(\theta_t + 1) \leq G(\theta_t) - \frac{\alpha_t}{n} \tau_t \left(1 - \frac{L\alpha_t\sigma^2}{2n} \right) \quad (12)$$

Next the authors, use of τ_t to show that $E[\tau_t] = 0 \Leftrightarrow \nabla G(\theta_t) = 0$. This fact is then used to prove the second part of the theorem.

5.2. Theorem 2

The following proof is used to determine the convergence rate of the method. If we assume G is Lipschitz-smooth with constant L and the training loss function f_i has σ -bounded gradients, it can be proven that $\mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \epsilon$ within $O(1/\epsilon^2)$ steps.

Beginning from the gradient descent update rule:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla G(\theta_t), \quad (13)$$

and $\alpha_t \leq \frac{2n}{L\sigma^2}$ as per the given conditions.

Since G is known to be Lipschitz-smooth,

$$G(\theta_{t+1}) \leq G(\theta_t) + \nabla G(\theta_t)^\top (\theta_{t+1} - \theta_t) + \frac{L}{2} \|\theta_{t+1} - \theta_t\|^2. \quad (14)$$

We can substitute the update rule into the inequality:

$$G(\theta_{t+1}) \leq G(\theta_t) - \alpha_t \|\nabla G(\theta_t)\|^2 + \frac{L\alpha_t^2}{2} \|\nabla G(\theta_t)\|^2. \quad (15)$$

Next we use the bound on α_t to solve for $\|\nabla G(\theta_t)\|^2$

$$\|\nabla G(\theta_t)\|^2 \leq \frac{2}{\alpha_t(2 - L\alpha_t)} (G(\theta_t) - G(\theta_{t+1})). \quad (16)$$

From there, we can take the expectation of both sides. After summing both sides of the equation over T steps and dividing by T , we are left with the following result:

$$\min_{0 \leq t < T} \mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\|\nabla G(\theta_t)\|^2], \quad (17)$$

Since the right side of the equation is independent of convergence, we can express it as a constant C , divided by \sqrt{T} . The \sqrt{T} represents the diminishing impact of each step on reducing the gradient norm as T increases.

$$\min_{0 \leq t < T} \mathbb{E}[\|\nabla G(\theta_t)\|^2] \leq \frac{C}{\sqrt{T}}. \quad (18)$$

Thus, it has been shown that the algorithm can converge to a constant ϵ in $O(1/\epsilon^2)$ steps.