FlexMatch: Boosting Semi-Supervised Learning with Curriculum Pseudo Labeling

Bowen Zhang*
Tokyo Institute of Technology
bowen.z.ab@m.titech.ac.jp

Yidong Wang*
Tokyo Institute of Technology
wang.y.ca@m.titech.ac.jp

Wenxin Hou
Microsoft
wenxinhou@microsoft.com

Hao Wu
Tokyo Institute of Technology
wu.h.aj@m.titech.ac.jp

Jindong Wang†
Microsoft Research Asia
jindwang@microsoft.com

Manabu Okumura†
Tokyo Institute of Technology
oku@pi.titech.ac.jp

Takahiro Shinozaki†
Tokyo Institute of Technology
shinot@ict.e.titech.ac.jp

Abstract

The recently proposed FixMatch achieved state-of-the-art results on most semi-supervised learning (SSL) benchmarks. However, like other modern SSL algorithms, FixMatch uses a pre-defined constant threshold for all classes to select unlabeled data that contribute to the training, thus failing to consider different learning status and learning difficulties of different classes. To address this issue, we propose Curriculum Pseudo Labeling (CPL), a curriculum learning approach to leverage unlabeled data according to the model's learning status. The core of CPL is to flexibly adjust thresholds for different classes at each time step to let pass informative unlabeled data and their pseudo labels. CPL does not introduce additional parameters or computations (forward or backward propagation). We apply CPL to FixMatch and call our improved algorithm *FlexMatch*. FlexMatch achieves state-of-the-art performance on a variety of SSL benchmarks, with especially strong performances when the labeled data are extremely limited or when the task is challenging. For example, FlexMatch outperforms FixMatch by **14.32**% and **24.55**% on CIFAR-100 and STL-10 datasets respectively, when there are only 4 labels per class. CPL also significantly boosts the convergence speed, e.g., FlexMatch can use only 1/5 training time of FixMatch to achieve even better performance. Furthermore, we show that CPL can be easily adapted to other SSL algorithms and remarkably improve their performances. We open source our code at https://github.com/TorchSSL/TorchSSL.

1 Introduction

Semi-supervised learning (SSL) has attracted increasing attention in recent years due to its superiority in leveraging a large amount of unlabeled data. This is particularly advantageous when the labeled data are limited in quantity or laborious to obtain. Consistency regularization [1–3] and pseudo labeling [4–8] are two powerful techniques for utilizing unlabeled data and have been widely used in modern SSL algorithms [9–13]. The recently proposed FixMatch [14] achieves competitive results

*Equal contribution.
†Corresponding author.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

by combining these techniques with weak and strong data augmentations and using cross-entropy loss as the consistency regularization criterion.

However, a drawback of FixMatch and other popular SSL algorithms such as Pseudo-Labeling [4] and Unsupervised Data Augmentation (UDA) [11] is that they rely on a *fixed* threshold to compute the unsupervised loss, using only unlabeled data whose prediction confidence is above the threshold. While this strategy can make sure that only high-quality unlabeled data contribute to the model training, it ignores a considerable amount of other unlabeled data, especially at the early stage of the training process, where only a few unlabeled data have their prediction confidence above the threshold. Moreover, modern SSL algorithms handle all classes *equally* without considering their different learning difficulties.

To address these issues, we propose Curriculum Pseudo Labeling (CPL), a curriculum learning [15] strategy to take into account the learning status of each class for semi-supervised learning. CPL substitutes the pre-defined thresholds with *flexible* thresholds that are dynamically adjusted for each class according to the current learning status. Notably, this process does not introduce any additional parameter (hyperparameter or trainable parameter) or extra computation (forward or back propagation). We apply this curriculum learning strategy directly to FixMatch and call the improved algorithm *FlexMatch*.

While the training speed remains as efficient as that of FixMatch, FlexMatch converges significantly faster and achieves state-of-the-art performances on most SSL image classification benchmarks. The benefit of introducing CPL is particularly remarkable when the labels are scarce or when the task is challenging. For instance, on the CIFAR-100 dataset, FlexMatch surpasses the performance of FixMatch by 14.32%, 4.30%, and 2.55% when the label amount is 400, 2500, and 10000 respectively. Moreover, CPL further shows its superiority by boosting the convergence speed – with CPL, FlexMatch takes less than 1/5 training time of FixMatch to reach its final accuracy. Adapting CPL to other modern SSL algorithms also leads to improvements in accuracy and convergence speed.

To sum up, this paper makes the following three contributions:

- We propose Curriculum Pseudo Labeling (CPL), a curriculum learning approach of dynamically leveraging unlabeled data for SSL. It is almost cost-free and can be easily integrated to other SSL methods.
- CPL significantly boosts the accuracy and convergence performance of several popular SSL algorithms on common benchmarks. Specifically, FlexMatch, the integration of FixMatch and CPL, achieves state-of-the-art results.
- We open-source TorchSSL, a unified PyTorch-based semi-supervised learning codebase for the fair study of SSL algorithms. TorchSSL includes implementations of popular SSL algorithms and their corresponding training strategies, and is easy to use or customize.

2 Background

Consistency regularization follows the continuity assumption of SSL [1, 2]. The most basic consistency loss in SSL, such as in Π Model [9], Mean Teacher [10] and MixMatch [12], is the ℓ-2 loss:

$$\sum_{b=1}^{\mu B} ||p_m(y|\omega(u_b)) - p_m(y|\omega(u_b))||_2^2, \tag{1}$$

where B is the batch size of labeled data, μ is the ratio of unlabeled data to labeled data, ω is a stochastic data augmentation function (thus the two terms in Eq.(1) are different), u_b denotes a piece of unlabeled data, and p_m represents the output probability of the model. With the introduction of pseudo labeling techniques [5, 7], the consistency regularization is converted to an entropy minimization process [16], which is more suitable for the classification task. The improved consistency loss with pseudo labeling can be represented as:

$$\frac{1}{\mu B} \sum_{b=1}^{\mu B} \mathbb{1}(\max(p_m(y|\omega(u_b))) > \tau) H(\hat{p}_m(y|\omega(u_b)), p_m(y|\omega(u_b))), \tag{2}$$

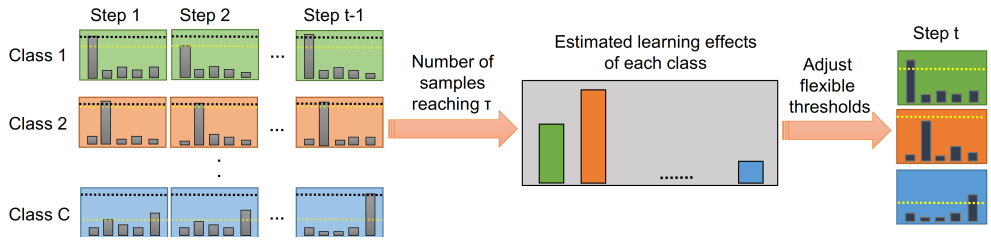


Figure 1: Illustration of Curriculum Pseudo Label (CPL). The estimated learning effects of each class are decided by the number of unlabeled data samples falling into this class and above the fixed threshold. They are then used to adjust the flexible thresholds to let pass the optimal unlabeled data. Note that the estimated learning effects do not always grow – they may also decrease if the predictions of the unlabeled data fall into other classes in later iterations.

where H is cross-entropy, τ is the pre-defined threshold and $\hat{p}_m(y|\omega(u_b))$ is the pseudo label that can either be a 'hard' one-hot label [4, 14] or a sharpened 'soft' one [11]. The intention of using a threshold is to mask out noisy unlabeled data that have low prediction confidence.

FixMatch utilizes such consistency regularization with strong augmentation to achieve competitive performance. For unlabeled data, FixMatch first uses weak augmentation to generate artificial labels. These labels are then used as the target of strongly-augmented data. The unsupervised loss term in FixMatch thereby has the form:

$$\frac{1}{\mu B}\sum_{b=1}^{\mu B} \mathbb{1}(\max(p_m(y|\omega(u_b))) > \tau) H(\hat{p}_m(y|\omega(u_b)), p_m(y|\Omega(u_b))), \tag{3}$$

where Ω is a strong augmentation function instead of weak augmentation ω.

Of the aforementioned works, the pre-defined threshold (τ) is constant. We believe this can be improved because the data of some classes may be inherently more difficult to learn than others. Curriculum learning [15] is a learning strategy where learning samples are gradually introduced according to the model's learning process. In such a way, the model is always optimally challenged. This technique is widely employed in deep learning research [17–21].

3 FlexMatch

3.1 Curriculum Pseudo Labeling

While current SSL algorithms render pseudo labels of only high-confidence unlabeled data cut off by a pre-defined threshold, CPL renders the pseudo labels *to different classes* and *at different time steps*. Such a process is realized by adjusting the thresholds according to the model's learning status of each class.

However, it is non-trivial to dynamically determine the thresholds according to the learning status. The most ideal approach would be calculating evaluation accuracies for each class and use them to scale the threshold, as:

$$\mathcal{T}_t(c) = a_t(c) \cdot \tau, \tag{4}$$

where $\mathcal{T}_t(c)$ is the flexible threshold for class c at time step t and $a_t(c)$ is the corresponding evaluation accuracy. In this way, lower accuracy that indicates a less satisfactory learning status of the class will lead to a lower threshold that encourages more samples of this class to be learned. Since we cannot use the evaluation set in the model learning process, one may have to separate an extra validation set from the training set for such accuracy evaluations. However, this practice show two fatal problems: First, such a *labeled* validation set separated from the training set is expensive under SSL scenario as the labeled data are already scarce. Second, to dynamically adjust the thresholds in the training process, accuracy evaluations must be done continually at each time step t, which will considerably slow down the training speed.

In this work, we propose Curriculum Pseudo Labeling (CPL) for semi-supervised learning. Our CPL uses an alternative way to estimate the learning status, which does not introduce additional inference processes, nor needs an extra validation set. As believed in [14], a high threshold that filters out noisy pseudo labels and leaves only high-quality ones can considerably reduce the confirmation bias [22]. Therefore, our key assumption is that when the threshold is high, the learning effect of a class can be reflected by the number of samples whose predictions fall into this class and above the threshold. Namely, the class with fewer samples having their prediction confidence reach the threshold is considered to have a greater learning difficulty or a worse learning status, formulated as:

$$\sigma_t(c) = \sum_{n=1}^{N} \mathbb{1}(\max(p_{m,t}(y|u_n)) > \tau) \cdot \mathbb{1}(\arg\max(p_{m,t}(y|u_n) = c). \tag{5}$$

where $\sigma_t(c)$ reflects the learning effect of class c at time step t. $p_{m,t}(y|u_n)$ is the model's prediction for unlabeled data u_n at time step t, and N is the total number of unlabeled data. When the unlabeled dataset is balanced (i.e., the number of unlabeled data belonging to different classes are equal or close), larger $\sigma_t(c)$ indicates a better estimated learning effect. By applying the following normalization to $\sigma_t(c)$ to make its range between 0 to 1, it can then be used to scale the fixed threshold τ:

$$\beta_t(c) = \frac{\sigma_t(c)}{\max_c \sigma_t}, \tag{6}$$

$$\mathcal{T}_t(c) = \beta_t(c) \cdot \tau. \tag{7}$$

One characteristic of such a normalization approach is that the best-learned class has its $\beta_t(c)$ equal to 1, causing its flexible threshold equal to τ. This is desirable. For classes that are hard to learn, the thresholds are lowered down, encouraging more training samples in these classes to be learned. This also improves the data utilization ratio. As learning proceeds, the threshold of a well-learned class is raised higher to selectively pick up higher-quality samples. Eventually, when all classes have reached reliable accuracies, the thresholds will all approach τ. Note that the thresholds do not always grow, it may also decrease if the unlabeled data is classified into a different class in later iterations. This new threshold is used for calculating the unsupervised loss in FlexMatch, which can be formulated as:

$$\mathcal{L}_{u,t} = \frac{1}{\mu B} \sum_{b=1}^{\mu B} \mathbb{1}(\max(q_b) > \mathcal{T}_t(\arg\max(q_b)))H(\hat{q}_b, p_m(y|\Omega(u_b))), \tag{8}$$

where $q_b = p_m(y|\omega(u_b))$. The flexible thresholds are updated at each iteration. Finally, we can formulate the loss in FlexMatch as the weighted combination (by λ) of supervised and unsupervised loss:

$$\mathcal{L}_t = \mathcal{L}_s + \lambda \mathcal{L}_{u,t}, \tag{9}$$

where $\mathcal{L}_s$ is the supervised loss on labeled data:

$$\mathcal{L}_s = \frac{1}{B} \sum_{b=1}^{B} H(y_b, p_m(y|\omega(x_b))). \tag{10}$$

Note that the cost of introducing CPL is almost free. Practically, every time the prediction confidence of an unlabeled data u_n is above the fixed threshold τ, the data, and its predicted class are marked and will be used for calculating $\beta_t(c)$ at the next time step. Such marking actions are bonus actions each time the consistency loss is computed. Therefore, FlexMatch does not introduce additional forward propagation processes for evaluating the model's learning status, nor new parameters.

3.2 Threshold warm-up

We noticed in our experiments that at the early stage of the training, the model may blindly predict most unlabeled samples into a certain class depending on the parameter initialization(i.e., more likely to have confirmation bias). Hence, the estimated learning status may not be reliable at this stage. Therefore, we introduce a warm-up process by rewriting the denominator in Eq. (6) as:

$$\beta_t(c) = \frac{\sigma_t(c)}{\max\left\{\max_c \sigma_t, N - \sum_c \sigma_t\right\}}, \tag{11}$$

Algorithm 1 FlexMatch algorithm.

Input: $\mathcal{X} = \{(x_m, y_m) : m \in (1, \dots, M)\}$, $\mathcal{U} = \{u_n : n \in (1, \dots, N)\}$ {M labeled data and N unlabeled data.}
$\hat{u}_n = -1 : n \in (1, \dots, N)$ {Initialize predictions of all unlabeled data as -1 indicating unused.}
while not reach the maximum iteration **do**
  for $c = 1$ to C **do**
    $\sigma(c) = \sum_{n=1}^{N} \mathbb{1}(\hat{u}_n = c)$ {Compute estimated learning effect.}
    if $\max \sigma(c) < \sum_{n=1}^{N} \mathbb{1}(\hat{u}_n = -1)$ **then**
      Calculate $\beta(c)$ using Eq. (11) {Threshold warms up when unused data dominate.}
    else
      Calculate $\beta(c)$ using Eq. (6) {Compute normalized estimated learning effect.}
    end if
    Calculate $\mathcal{T}(c)$ using Eq. (7) {Determine the flexible threshold for class c.}
  end for
  for $b = 1$ to μB **do**
    if $p_m(y|\omega(u_b)) > \tau$ **then**
      $\hat{u}_b = \arg\max q_b$ {Update the prediction of unlabeled data u_b.}
    end if
  end for
  Compute the loss via Eq. (8), (10) and (9).
end while
Return: Model parameters.

where the term $N - \sum_{c=1}^{C} \sigma_t(c)$ can be regarded as the number of unlabeled data that have not been used. This ensures that at the beginning of the training, all estimated learning effects gradually rise from 0 until the number of unused unlabeled data is no longer predominant. The duration of such a period depends on the unlabeled data amount (ref. N in Eq. (11)) and the learning difficulty (ref. the growing speed of $\sigma_t(c)$ in Eq. (11)) of the dataset. In practice, such a warm-up process is very easy to implement as we can add an extra class to denote the unused unlabeled data. Thus calculating the denominator of Eq. (11) is simply converted to finding the maximum among $c + 1$ classes.

3.3 Non-linear mapping function

The flexible threshold in Eq. (7) is determined by the normalized estimated learning effects via a linear mapping. However, it may not be the most suitable mapping in the real training process, where the increase or decrease of $\beta_t(c)$ may make big jumps in the early phase where the predictions of the model are still unstable; and only make small fluctuations after the class is well-learned in the mid and late training stage. Therefore, it is preferable if the flexible thresholds can be more sensitive when $\beta_t(c)$ is large and vice versa.

We propose a non-linear mapping function to enable the thresholds to have a non-linear increasing curve when $\beta_t(c)$ ranges uniformly from 0 to 1, as formulated below:

$$\mathcal{T}_t(c) = \mathcal{M}(\beta_t(c)) \cdot \tau, \tag{12}$$

where $\mathcal{M}(\cdot)$ is a non-linear mapping function. It is clear that Eq. (7) can be seen as a special case by setting $\mathcal{M}$ to the identity function. The mapping function $\mathcal{M}$ should be monotonically increasing and have a maximum no larger than $1/\tau$ (otherwise the flexible threshold can be larger than 1 and filter out all samples). To avoid introducing additional hyperparameters (e.g. lower limits of the flexible thresholds), we consider the mapping function to have a range from 0 to 1 so that the flexible thresholds range from 0 to τ.

A monotone increasing convex function lets the thresholds grow slowly when $\beta_t(c)$ is small, and become more sensitive as $\beta_t(c)$ gets larger. Hence, we intuitively choose a convex function with the above-mentioned properties $\mathcal{M}(x) = \frac{x}{2-x}$ for our experiments. We also conduct an ablation study to compare among mapping functions with different convexity and concavity in Sec. 4.4. The full algorithm of FlexMatch is shown in Algorithm 1.

Table 1: Error rates on CIFAR-10/100, SVHN, and STL-10 datasets. The 'Flex' prefix denotes applying CPL to the algorithm, and 'PL' is an abbreviation of Pseudo-Labeling. STL-10 dataset does not have label information for unlabeled data, thus its fully-supervised result is unavailable.

Dataset	CIFAR-10			CIFAR-100			STL-10			SVHN	
Label Amount	40	250	4000	400	2500	10000	40	250	1000	40	1000
PL	69.51$\pm$4.55	41.02$\pm$3.56	13.15$\pm$1.84	86.10$\pm$1.50	58.00$\pm$0.38	36.48$\pm$0.13	74.48$\pm$1.48	55.63$\pm$5.38	31.80$\pm$0.29	60.32$\pm$2.46	9.56$\pm$0.25
Flex-PL	**65.41**$\pm$1.35	**36.37**$\pm$1.57	**10.82**$\pm$0.04	**74.85**$\pm$1.53	**44.15**$\pm$0.19	**29.13**$\pm$0.26	**69.26**$\pm$0.60	**41.28**$\pm$0.46	**24.63**$\pm$0.14	**36.90**$\pm$1.19	**8.64**$\pm$0.08
UDA	7.33$\pm$2.03	5.11$\pm$0.07	4.20$\pm$0.12	44.99$\pm$2.28	27.59$\pm$0.24	22.09$\pm$0.19	37.31$\pm$3.03	12.07$\pm$1.50	6.65$\pm$0.25	4.40$\pm$2.31	**1.93**$\pm$0.01
Flex-UDA	**5.33**$\pm$0.13	**5.05**$\pm$0.02	**4.07**$\pm$0.06	**33.64**$\pm$0.92	**24.34**$\pm$0.20	**20.07**$\pm$0.13	**12.84**$\pm$2.60	**8.05**$\pm$0.21	**5.77**$\pm$0.08	**3.78**$\pm$1.67	1.97$\pm$0.06
FixMatch	6.78$\pm$0.50	4.95$\pm$0.07	4.09$\pm$0.02	46.76$\pm$0.79	28.15$\pm$0.81	22.47$\pm$0.66	35.42$\pm$6.43	10.49$\pm$1.03	6.20$\pm$0.20	**4.36**$\pm$2.16	**1.97**$\pm$0.03
FlexMatch	**4.99**$\pm$0.16	**4.80**$\pm$0.06	**3.95**$\pm$0.03	**32.44**$\pm$1.99	**23.85**$\pm$0.23	**19.92**$\pm$0.06	**10.87**$\pm$1.15	**7.71**$\pm$0.14	**5.56**$\pm$0.22	5.36$\pm$2.38	2.86$\pm$0.91
Fully-Supervised	4.45$\pm$ 0.12			19.07$\pm$ 0.18			-			2.14$\pm$ 0.02	

4 Experiments

We evaluate FlexMatch and other CPL-enabled algorithms on common SSL datasets: CIFAR-10/100 [23], SVHN [24], STL-10 [25] and ImageNet [26], and extensively investigate the performance under various labeled data amounts. We mainly compare our method with Pseudo-Labeling [4], UDA [11] and FixMatch [14], since they all involve a pre-defined threshold. The results of other popular SSL algorithms are in the appendix B. We also add a fully-supervised experiment for each dataset to better understand the results of SSL algorithms. Note that previously suggested [27] fully-supervised comparisons use only the labeled set for training, whose purpose is to manifest the improvement brought by the introduction of unlabeled data. With the development of modern SSL algorithms, however, semi-supervised approaches are achieving competitive performance with supervised ones, or even better performance due to the strength of consistency regularization. Therefore, our fully-supervised comparisons are conducted with all data labeled, and apply weak data augmentations following Eq. (10). We re-implement all baselines using our PyTorch [28] codebase: TorchSSL, which is introduced in the appendix B.

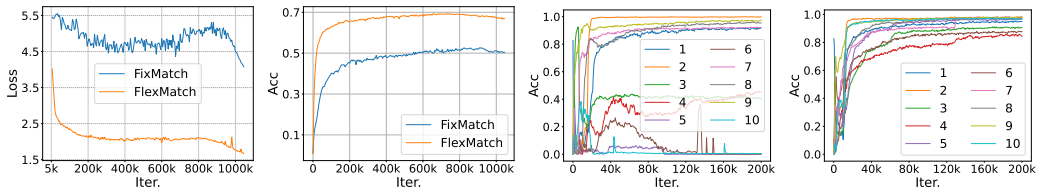
For a fair comparison, we use the same hyperparameters following FixMatch [14]. Concretely, the optimizer for all experiments is standard stochastic gradient descent (SGD) with a momentum of 0.9 [29, 30]. For all datasets, we use an initial learning rate of 0.03 with a cosine learning rate decay schedule [31] as $\eta = \eta_0 \cos(\frac{7\pi k}{16K})$, where η_0 is the initial learning rate, k is the current training step and K is the total training step that is set to 2^{20}. We also perform an exponential moving average with the momentum of 0.999. The batch size of labeled data is 64 except for ImageNet. μ is set to be 1 for Pseudo-Label and 7 for UDA, FixMatch, and FlexMatch. τ is set to 0.8 for UDA and 0.95 for Pseudo Label, FixMatch, and FlexMatch. These setups follow the original papers. The strong augmentation function used in our experiments is RandAugment [32]. We use ResNet-50 [33] for the ImageNet experiment and Wide ResNet (WRN) [34] and its variant [35] for other datasets. Detailed hyperparameters are listed in the appendix A.

We adopt two evaluation metrics: (1) the median error rate of the last 20 checkpoints following [12, 14], and (2) the best error rate in all checkpoints. We argue that the median approach is not suitable when the convergence speeds of the algorithms show significant differences – the large number of redundant iterations may result in over-fitting for the fast-converge algorithms. Therefore, we report the best error rates for all algorithms, while the results of the median approach are also provided in the appendix A, showing that our FlexMatch still achieves the best performance. We run each task three times using distinct random seeds to obtain the error bars.

4.1 Main results

The classification error rates on CIFAR-10/100, STL-10 and SVHN datasets are in Table 1[3], and the results on ImageNet are in Sec. 4.2. Note that the SVHN dataset used in our experiment also includes the extra set that contains 531,131 additional samples. Results demonstrate that FlexMatch achieves the best performance under all label conditions on all datasets except for SVHN where Flex-UDA (i.e., UDA with CPL) and UDA have the lowest error rate on the 40-label split and the 1000-label split, respectively. We also provide the detailed precision, recall, F1, and AUC results in the appendix A. Our CPL (FlexMatch) has the following advantages:

[3]The latest result will be updated on Github which may differ from the table.



(a) Loss (b) Top-1 acc. (c) FixMatch: 56.4% (d) FlexMatch: 94.3%

Figure 3: Convergence analysis of FixMatch and FlexMatch. (a) and (b) depict the loss and top-1-accuracy on CIFAR-100 with 400 labels. Evaluations are done every 5K iterations. (c) and (d) demonstrate the class-wise accuracy within the first 200K iterations on CIFAR-10 dataset. The numbers in legend correspond to the ten classes in the dataset.

CPL achieves better performance on tasks with extremely limited labeled data. Our FlexMatch significantly outperforms other methods when the amount of labels is extremely small. For instance, on the CIFAR-100 dataset with 400 labels (i.e., only 4 label samples per class), FlexMatch achieves an average error rate of **32.44**%, which significantly outperforms FixMatch by **14.32**%.

CPL improves the performance of existing SSL algorithms. Other than FixMatch, the error rate of Pseudo-Labeling and UDA are also dramatically reduced by employing CPL. For instance, the error rate is reduced by **24.53**% for Pseudo-Labeling on SVHN (40 labels), and by **24.47**% for UDA on STL-10 (40 labels). These results further prove the effectiveness of CPL in better leveraging unlabeled data. Figure 2 shows the average running time of a single iteration with or without adding our CPL, it is clear that while improving the performance of existing SSL algorithms, our CPL *does not* introduce additional computational burden.

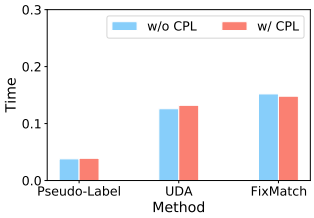


Figure 2: Average running time of one iteration on a single GeForce RTX 3090 GPU.

CPL achieves better performance on complicated tasks. The STL-10 dataset contains unlabeled data from a similar but broader distribution of images than its labeled set. The existence of new types of objects in the unlabeled dataset makes STL-10 a more challenging and realistic task. Our FlexMatch achieves even greater performance improvement under such a challenging situation. To the best of our knowledge, we are the first to use only 40 labeled data for STL-10, and obtain a *noteworthy* error rate of **10.87**%, which is substantially better than FixMatch (35.42%). Similar strong improvements are also observed on CIFAR-100 dataset, which has as many as one hundred classes.

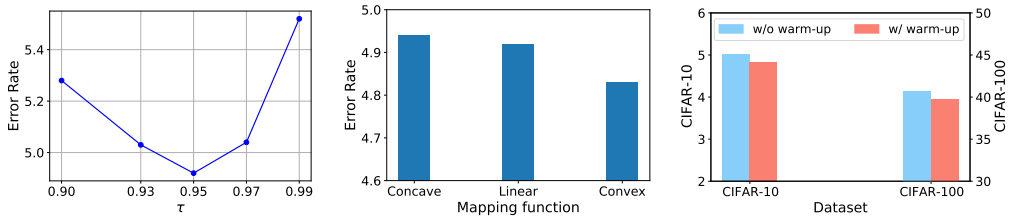
We also analyze the reason why FlexMatch fails to surpass FixMatch on SVHN. This is probably because SVHN is a relatively *simple* (i.e., to classify digits) yet *unbalanced* dataset. The class-wise imbalance leads to the classes with fewer samples never have their estimated learning effects close to 1 according to Eq. (6), even when they are already well-learned. Such low thresholds allow noisy pseudo-labeled samples to be trusted and learned throughout the training process, which is also reflected by the loss descent curve where the low-threshold classes have major fluctuations. FixMatch, on the other hand, fixes its threshold at 0.95 to filter out noisy samples. Such a fixed high threshold is not preferable with respect to both accuracies of hard-to-learn classes and overall convergence speed as explained earlier, but since SVHN is an easy task, the model can easily learn the task and make high-confidence predictions, setting a high-fixed threshold thus becomes less problematic and has its advantages overweighed.

4.2 Results on ImageNet

We also verify the effectiveness of CPL on ImageNet-1K [26] which is a much more realistic and complicated dataset. We randomly choose the same 100K labeled data (i.e., 100 labels per class), which is less than 8% of the total labels. We keep most of the hyperparameters consistent with on other datasets except for a different batch size and weight decay value, and show the error rate comparison after

Table 2: Error rate results on ImageNet after 2^{20} iterations.

Method	Top-1	Top-5
FixMatch	43.08	19.55
FlexMatch	**35.21**	**13.96**



(a) Threshold τ (b) Mapping function (c) Threshold warm-up

Figure 4: Ablation study of FlexMatch.

running 2^{20} iterations in Table 2. With all hyperparameters being the same, by only introducing CPL to FixMatch, the error rates are significantly reduced, further proving the effectiveness of CPL on challenging tasks. This result also indicates that when the task is complicated, despite the class imbalance issue (the number of images within each class ranges from 732 to 1300), CPL can still bring considerable improvements. Note that this result does not represent the best performance of each algorithm as the model cannot fully converge after 2^{20} iterations, and due to the computational resource limitation, we did not deliberately tune the hyperparameters to obtain the best result for FlexMatch on ImageNet.

4.3 Convergence speed acceleration

Another strong advantage of FlexMatch is its superior convergence speed. Figure 3(a) and 3(b) shows the comparison between FlexMatch and FixMatch with respect to the loss and top-1-accuracy on CIFAR-100 400-label split. The loss of FlexMatch decreases much faster and smoother than FixMatch, demonstrating its superior convergence speed. The major fluctuations of the loss in FixMatch may due to the pre-defined threshold that lets pass most unlabeled data belonging to certain classes, whereas with CPL a larger batch of unlabeled data containing samples from various classes enables the gradient to more directly head toward the global optimum. As a result, with only 50K iterations, FlexMatch has already surpassed the final results of FixMatch. After 800K iterations, however, we observe a further decrease in loss and accuracy. This is likely due to over-fitting, which also occurs in FixMatch after 900K iterations. Thus, we believe it is not fair to use the median results of the last few checkpoints for evaluating algorithms with different convergence speeds.

We further compare the class-wise accuracy of FixMatch and FlexMatch on CIFAR-10 in their early training stages. As shown in Figure 3(c) and 3(d), at iteration 200K, FixMatch only hits an overall accuracy of 56.35% as half of the classes are still learned unsatisfactorily, whereas FlexMatch has already achieved an overall accuracy of 94.29% which is even higher than the final accuracy reached by FixMatch after 1M iterations. It is manifest that the introduction of CPL successfully encourages the model to proactively learn those difficult classes thereby improving the overall learning effect.

4.4 Ablation study

We conduct experiments to evaluate three components of FlexMatch: the upper limit of thresholds τ, mapping functions $\mathcal{M}(x)$, and threshold warm-up.

Threshold upper bound. We investigate 5 different τ values and 3 different mapping functions on CIFAR-10 dataset with 40 labels. As shown in Figure 4(a), the optimal choice of τ is around 0.95, either increasing or decreasing this value results in a performance decay. Note that in FlexMatch, tuning τ does not only affect the upper limit of the threshold but also the estimated learning effects because they are determined by the number of samples that fall above τ.

Mapping function. We explore three different mapping functions in Figure 4(b): (1) concave: $\mathcal{M}(x) = \ln(x+1)/\ln 2$, (2) linear: $\mathcal{M}(x) = x$, and (3) convex: $\mathcal{M}(x) = x/(2-x)$. We see that the convex function shows the best performance and the concave function shows the worst. Although tweaking the degree of convexity may probably lead to further improvement, we do not make further investigation in this paper. It is noteworthy that all these functions have their outputs grow from 0 to 1 when the inputs go from 0 to 1. One may also design a function with a different range, for instance, from 0.5 to 1. In this case, it is equivalent to setting a lower limit to the flexible threshold so that even

at the beginning of the training, only samples with prediction confidence higher than this limit will contribute to the unsupervised loss. We do not include such a lower limit in FlexMatch since it will introduce a new hyperparameter. However, we did find that setting a lower limit at 0.5 can slightly improve the performance. A possible reason is that the lower threshold prevents noisy training caused by incorrect pseudo labels at the early stage [36].

Threshold warm-up. We analyze the performance of threshold warm-up on both CIFAR-10 (40 labels) and CIFAR-100 (400 labels) datasets. As shown in Figure 4(c), the improvement brought by threshold warm-up is about 0.2% on CIFAR-10 and about 1% on CIFAR-100. This is reasonable since CIFAR-100 is a more complicated dataset with more classes compared to CIFAR-10. At the beginning of the training without the threshold warm-up, the flexible thresholds may go through heavy fluctuations because the denominator in Eq.(6) is small. In the meantime, there will always be some classes whose flexible thresholds reach or approach τ, thereby filtering out most unlabeled data in the batch. The threshold warm-up solves this issue by gradually raising the thresholds of all classes from zero – it creates a learning boom at the early training stage where most of the unlabeled data can be utilized.

Comparison with class balancing objectives. CPL has the effect of balancing across classes the number of unlabeled samples used to compute pseudo-labeling loss in each batch. Similar effect can be achieved by making the marginal class distribution close to a uniform distribution for each batch. We conduct such a comparative experiment by directly adding an additional objective to FixMatch: $\mathcal{L}_b = \sum_c q_c \log(q_c/\hat{p}_c)$ [22], where $\hat{p}_c$ is the mean predicted probability of class c across all samples in the batch, and q is a uniform distribution: $q_c = 1/C$. Unfortunately, the error rate of adding such an objective is 7.16% on the CIFAR-10 40-label split (compared with FixMatch 6.78%$\pm$0.5), which does not imply any accuracy improvement. While this approach requires instances of each class within each batch to be balanced to make sense, CPL does not have such a constraint. It is more flexible and involves less human intervention to adjust thresholds than adjusting model's predictions.

5 Related Work

Pseudo-Labeling [4] is a pioneer SSL method that uses hard artificial labels converted from model predictions. A confidence-based strategy was used in [6] along with pseudo labeling so that the unlabeled data are used only when the predictions are sufficiently confident. Such confidence-based thresholding also presents in recently proposed UDA [11] and FixMatch [14] with the difference being that UDA used sharpened 'soft' pseudo labels with a temperature whereas Fixmatch adopted one-hot 'hard' labels. The success of UDA and FixMatch, however, relies heavily on the usage of strong data augmentations to improve the consistency regularization. ReMixMatch [13] also leveraged such strong augmentations.

The combination of curriculum learning and semi-supervised learning is popular in recent years [37–39]. For multi-model image classification task, [37] optimized the learning process of unlabeled images by judging their reliability and discriminability. In [38], the easy image-level properties are learned first and then used to facilitate segmentation via constrained CNNs. Curriculum learning is also used to alleviate out-of-distribution problems by picking up in-distribution samples from unlabeled data according to the out-of-distribution scores [39].

Several researches have investigated on dynamic threshold in related fields such as sentiment analysis [40] and semantic segmentation [41]. In [40], the threshold was gradually reduced to make high-quality data selected into labeled data set in the early stage and large-quantity in the later stage. An extra classifier is added to automate the threshold to deal with domain inconsistency in [41]. [42] introduced curriculum learning to self-training with a steadily increasing threshold and achieved near state-of-the-art results.

6 Conclusion and Future Work

In this paper, we introduce Curriculum Pseudo Labeling (CPL), a curriculum learning approach of leveraging unlabeled data for SSL. CPL dramatically improves the performance and convergence speed of SSL algorithms that involve thresholds while being extremely simple and almost cost-free. FlexMatch, our improved algorithm of FixMatch, achieves state-of-the-art performance on a variety

of SSL benchmarks. In future work, we would like to improve our method under the long-tail scenario where the unlabeled data belonging to each class are extremely unbalanced.

Broader Impact

CPL fills the gap that no modern SSL algorithm considers the inherent learning difficulties of different classes during the training, and shows that by doing so, the convergence speed and final accuracy can both be improved. We hope that CPL can attract more future attention to explore the effectiveness of utilizing unlabeled data according to the model's learning status as well as the per-class learning difficulty.

Funding Disclosure

Funding in direct support of this work: computing resource granted by Tokyo Institute of Technology and Microsoft Research Asia.

References

[1] Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles. In *NeurIPS*, pages 3365–3373, 2014.

[2] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *ICLR*, 2017.

[3] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *NeurIPS*, pages 1171–1179, 2016.

[4] Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, 2013.

[5] Geoffrey J McLachlan. Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. *Journal of the American Statistical Association*, 70(350):365–369, 1975.

[6] Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. Semi-supervised self-training of object detection models. 2005.

[7] Henry Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3):363–371, 1965.

[8] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *CVPR*, pages 10687–10698, 2020.

[9] Antti Rasmus, Harri Valpola, Mikko Honkala, Mathias Berglund, and Tapani Raiko. Semi-supervised learning with ladder networks. In *NeurIPS*, pages 3546–3554, 2015.

[10] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NeurIPS*, pages 1195–1204, 2017.

[11] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *NeurIPS*, 33, 2020.

[12] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *NeurIPS*, page 5050–5060, 2019.

[13] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution matching and augmentation anchoring. In *ICLR*, 2019.

[14] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *NeurIPS*, 33, 2020.

[15] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.

[16] Yves Grandvalet, Yoshua Bengio, et al. Semi-supervised learning by entropy minimization. In *CAP*, pages 281–296, 2005.

[17] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. Curriculum learning of multiple tasks. In *CVPR*, pages 5492–5500, 2015.

[18] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander Hauptmann. Self-paced curriculum learning. In *AAAI*, volume 29, 2015.

[19] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *ICML*, pages 1311–1320. PMLR, 2017.

[20] Guy Hacohen and Daphna Weinshall. On the power of curriculum learning in training deep networks. In *ICML*, pages 2535–2544. PMLR, 2019.

[21] Daphna Weinshall, Gad Cohen, and Dan Amir. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *ICML*, pages 5238–5246. PMLR, 2018.

[22] Eric Arazo, Diego Ortego, Paul Albert, Noel E O'Connor, and Kevin McGuinness. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *IJCNN*, pages 1–8, 2020.

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

[24] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[25] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, pages 215–223, 2011.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009.

[27] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D Cubuk, and Ian J Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In *NeurIPS*, pages 3239–3250, 2018.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *NeurIPS*, 32:8026–8037, 2019.

[29] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, pages 1139–1147. PMLR, 2013.

[30] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[31] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *ICLR*, 2016.

[32] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[33] 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.

[34] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[35] Tianyi Zhou, Shengjie Wang, and Jeff Bilmes. Time-consistent self-supervision for semi-supervised learning. In *ICML*, pages 11523–11533. PMLR, 2020.

[36] Mamshad Nayeem Rizve, Kevin Duarte, Yogesh S Rawat, and Mubarak Shah. In defense of pseudo-labeling: An uncertainty-aware pseudo-label selection framework for semi-supervised learning. *arXiv preprint arXiv:2101.06329*, 2021.

[37] Chen Gong, Dacheng Tao, Stephen J Maybank, Wei Liu, Guoliang Kang, and Jie Yang. Multi-modal curriculum learning for semi-supervised image classification. *IEEE Transactions on Image Processing*, 25(7):3249–3260, 2016.

[38] Hoel Kervadec, Jose Dolz, Éric Granger, and Ismail Ben Ayed. Curriculum semi-supervised segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 568–576. Springer, 2019.

[39] Qing Yu, Daiki Ikami, Go Irie, and Kiyoharu Aizawa. Multi-task curriculum framework for open-set semi-supervised learning. In *ECCV*, pages 438–454. Springer, 2020.

[40] Yue Han, Yuhong Liu, and Zhigang Jin. Sentiment analysis via semi-supervised learning: a model based on dynamic threshold and multi-classifiers. *Neural Computing and Applications*, 32(9):5117–5129, 2020.

[41] Zhedong Zheng and Yi Yang. Rectifying pseudo label learning via uncertainty estimation for domain adaptive semantic segmentation. *IJCV*, 129(4):1106–1120, 2021.

[42] Paola Cascante-Bonilla, Fuwen Tan, Yanjun Qi, and Vicente Ordonez. Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6912–6920, 2021.

[43] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. fixmatch. `https://github.com/google-research/fixmatch`, 2020.

[44] Lee Doyup and Cheon Yeongjae. Fixmatch-pytorch. `https://github.com/LeeDoYup/FixMatch-pytorch`, 2020.

[45] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE TPAMI*, 41(8):1979–1993, 2018.

[46] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *CVPR*, pages 7151–7160, 2018.

A Experimental Results

A.1 Hyperparameter setting

For reproduction, we show the detailed hyperparameter setting for each method in Table 3 and 4, for algorithm-dependent and algorithm-independent hyperparameters, respectively.

Table 3: Algorithm dependent parameters.

Algorithm	PL (Flex-PL)	UDA (Flex-UDA)	FixMatch (FlexMatch)
Unlabeled Data to Labeled Data Ratio	1	7	7
Pre-defined Threshold	0.95	0.8	0.95
Temperature	-	0.5	-

Table 4: Algorithm independent parameters.

Dataset	CIFAR-10	CIFAR-100	STL-10	SVHN	ImageNet
Model	WRN-28-2 [34]	WRN-28-8	WRN-37-2 [35]	WRN-28-2	ResNet-50 [33]
Weight Decay	5e-4	1e-3	5e-4	5e-4	3e-4
Batch Size	64				32
Learning Rate	0.03				
SGD Momentum	0.9				
EMA Momentum	0.999				
Unsupervised Loss Weight	1				

A.2 Class-wise accuracy improvement.

As introduced in the paper, CPL has its ability of improving performance on those hard-to-learn classes by taking into consider the model's learning status. A detailed class-wise accuracy comparison is listed in Table 5, where the final accuracies of class 2, 3 and 5 with originally bad performance are improved.

Table 5: Class-wise accuracy comparison on CIFAR-10 40-label split.

Class Number	0	1	2	3	4	5	6	7	8	9
FixMatch	0.961	0.975	**0.785**	**0.851**	0.972	**0.874**	0.985	0.976	0.978	0.975
FlexMatch	0.969	0.984	**0.916**	**0.863**	0.966	**0.883**	0.984	0.976	0.981	0.973

A.3 Median error rates

We also report the median error rates of the last 20 checkpoints by allowing all methods to run the same iterations, following existing work [14]. There are 1000 iterations between every two checkpoints. The results in Table 6 show that our CPL method can dramatically improve the performance of existing SSL algorithms and the FlexMatch achieves the best accuracy. These conclusions are in consistency with the results of Table1 in the main text, showing the effectiveness of our proposed CPL algorithm.

Table 6: Median error rates of the last 20 checkpoints.

Dataset	CIFAR-10			CIFAR-100			STL-10			SVHN	
Label Amount	40	250	4000	400	2500	10000	40	250	1000	40	1000
PL	71.99$\pm$4.24	41.42$\pm$4.17	13.62$\pm$1.95	89.70$\pm$1.48	58.52$\pm$0.34	37.61$\pm$0.10	76.19$\pm$0.58	57.28$\pm$5.77	33.09$\pm$0.33	64.13$\pm$1.74	**9.98**$\pm$0.27
Flex-PL	**69.01**$\pm$4.95	**39.97**$\pm$9.32	**11.41**$\pm$0.07	**75.96**$\pm$1.64	**44.71**$\pm$0.30	**30.01**$\pm$0.16	**71.67**$\pm$0.94	**42.88**$\pm$0.17	**25.76**$\pm$0.31	**40.59**$\pm$1.54	11.31$\pm$0.75
UDA	7.47$\pm$1.89	**5.39**$\pm$0.11	4.46$\pm$0.17	54.04$\pm$3.45	29.73$\pm$0.16	23.52$\pm$0.27	48.69$\pm$8.03	13.13$\pm$2.90	8.24$\pm$1.02	4.83$\pm$2.57	**2.00**$\pm$0.03
Flex-UDA	**5.92**$\pm$0.35	5.41$\pm$0.14	**4.31**$\pm$0.05	**38.14**$\pm$1.18	**26.15**$\pm$0.18	**21.18**$\pm$0.22	**14.01**$\pm$3.34	**9.80**$\pm$0.09	**6.60**$\pm$0.03	**4.40**$\pm$2.31	2.08$\pm$0.07
FixMatch	7.41$\pm$1.13	5.22$\pm$0.07	4.30$\pm$0.07	50.36$\pm$0.80	28.95$\pm$0.11	22.98$\pm$0.53	49.71$\pm$7.91	13.08$\pm$3.03	7.19$\pm$0.66	**4.43**$\pm$2.19	**2.05**$\pm$0.04
FlexMatch	**5.66**$\pm$0.49	**5.03**$\pm$0.12	**4.23**$\pm$0.08	**35.61**$\pm$2.41	**25.42**$\pm$0.17	**21.20**$\pm$0.12	**13.03**$\pm$2.05	**9.01**$\pm$0.38	**6.02**$\pm$0.32	9.85$\pm$0.72	7.83$\pm$0.36

A.4 Detailed results

To comprehensively evaluate the performance of all methods in a classification setting, we further report the precision, recall, f1 score and AUC (area under curve) results on CIFAR-10 dataset. As shown in Table 7, we see that in addition to the reduced error rates, CPL also has the best performance on precision, recall, F1 score, and AUC. These metrics, together with error rates (accuracy), shows the strong performance of our proposed method.

Table 7: Precision, recall, f1 score and AUC results on CIFAR-10.

Label Amount	40 labels				4000 labels			
Criteria	Precision	Recall	F1 Score	AUC	Precision	Recall	F1 score	AUC
PL	0.2742	0.2766	0.2499	0.6685	0.8536	0.8536	0.8533	**0.9851**
Flex-PL	**0.2750**	**0.2846**	**0.2677**	**0.6712**	**0.8560**	**0.8567**	**0.8561**	0.9847
UDA	0.9018	0.8820	0.8770	0.9863	0.9557	0.9559	0.9557	0.9985
Flex-UDA	**0.9480**	**0.9483**	**0.9480**	**0.9974**	**0.9572**	**0.9573**	**0.9571**	**0.9985**
Fixmatch	0.9382	0.9369	0.9363	0.9942	0.9470	0.9472	0.9469	0.9980
Flexmatch	**0.9505**	**0.9506**	**0.9504**	**0.9974**	**0.9581**	**0.9581**	**0.9581**	**0.9984**

B TorchSSL: A PyTorch-based SSL Codebase

The PyTorch [28] framework has gained increasing attention in the deep learning research community. However, the main existing SSL codebase [43] is based on TensorFlow. For the convenience and customizability, we re-implement and open source a PyTorch-based SSL toolbox, named *TorchSSL* [4] as shown in Figure 5. TorchSSL contains eight popular semi-supervised learning methods: Π-Model [9], Pseudo-Labeling [4], VAT [45], Mean Teacher [10], MixMatch [12], ReMixMatch [13], UDA [11], and FixMatch [14], along with our proposed method FlexMatch. Most of our implementation details are based on [43]. More importantly, in addition to the basic SSL methods and components, we implement several techniques to make the results stable under PyTorch framework. For instance, we add synchronized batch normalization [46] to avoid the performance degradation caused by multi-GPU training with small batch size, and a batch norm controller to prevent performance crashes for some algorithms, which is not officially supported in PyTorch.

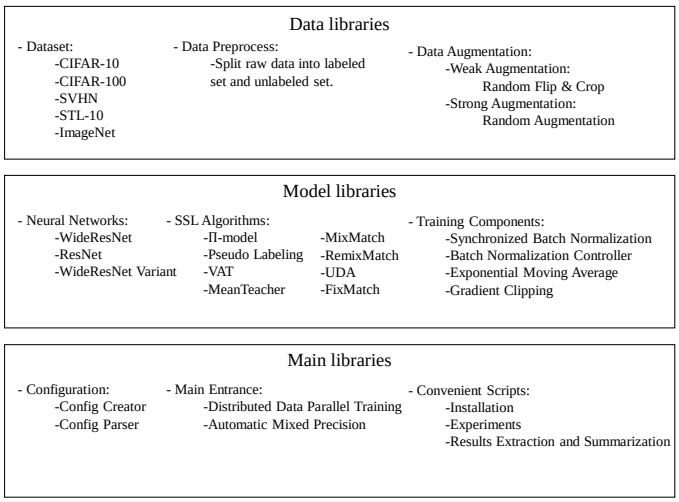


Figure 5: Components of TorchSSL.

[4] Our toolbox is partially based on [44].

B.1 BatchNorm Controller

We observed that Mean Teacher can be very unstable if we update BatchNorm for both labeled data and unlabeled data in turn. Other algorithms such as Π-Model and MixMatch also show the similar instability. Therefore, we use BatchNorm Controller to update BatchNorm only for labeled data if labeled data and unlabeled data are forwarded separately. The code of BatchNorm Controller is as follows. We record the BatchNorm statistics before the forward propagation of unlabeled data and restore them after the propagation is done.

B.2 Benchmark results

We comprehensively run all algorithms in our TorchSSL on four common datasets in SSL: CIFAR-10, CIFAR-100, SVHN, and STL-10, and report the best error rates in Table 8, 9, 10, and 11, respectively. These benchmark results provide a reference of using this toolbox.

Table 8: Benchmark results on CIFAR-10. The error bars are obtained from three trials.

Algorithms	Error Rate (40 labels)	Error Rate (250 labels)	Error Rate (4000 labels)
Π-Model [9]	$74.97_{\pm 1.80}$	$48.20_{\pm 0.68}$	$13.21_{\pm 0.40}$
Pseudo-Labeling [4]	$69.51_{\pm 4.55}$	$41.02_{\pm 3.56}$	$13.15_{\pm 1.84}$
VAT [45]	$76.40_{\pm 2.47}$	$44.58_{\pm 1.55}$	$11.31_{\pm 0.35}$
Mean Teacher [10]	$70.09_{\pm 4.28}$	$35.88_{\pm 5.00}$	$8.25_{\pm 0.10}$
MixMatch [12]	$36.33_{\pm 8.88}$	$14.51_{\pm 2.31}$	$6.81_{\pm 0.08}$
ReMixMatch [13]	$24.38_{\pm 9.32}$	$9.92_{\pm 4.74}$	$4.98_{\pm 0.11}$
UDA [11]	$7.33_{\pm 2.03}$	$5.11_{\pm 0.07}$	$4.18_{\pm 0.10}$
FixMatch [14]	$6.78_{\pm 0.50}$	$4.95_{\pm 0.07}$	$4.09_{\pm 0.02}$
FlexMatch	$4.99_{\pm 0.16}$	$4.80_{\pm 0.06}$	$3.95_{\pm 0.03}$

Table 9: Benchmark results on CIFAR-100.

Algorithms	Error Rate (400 labels)	Error Rate (2500 labels)	Error Rate (10000 labels)
Π-Model [9]	$85.90_{\pm 1.42}$	$58.72_{\pm 0.16}$	$36.36_{\pm 0.26}$
Pseudo-Labeling [4]	$86.10_{\pm 1.50}$	$58.00_{\pm 0.38}$	$36.48_{\pm 0.13}$
VAT [45]	$82.12_{\pm 0.02}$	$48.08_{\pm 0.48}$	$32.30_{\pm 0.13}$
Mean Teacher[10]	$78.66_{\pm 1.80}$	$45.48_{\pm 1.90}$	$31.44_{\pm 0.02}$
MixMatch [12]	$64.80_{\pm 1.90}$	$40.05_{\pm 0.33}$	$27.14_{\pm 0.16}$
ReMixMatch [13]	$43.39_{\pm 2.44}$	$26.54_{\pm 0.38}$	$20.28_{\pm 0.20}$
UDA [11]	$44.99_{\pm 2.28}$	$27.59_{\pm 0.24}$	$22.09_{\pm 0.19}$
FixMatch [14]	$46.76_{\pm 0.79}$	$28.15_{\pm 0.81}$	$22.47_{\pm 0.66}$
FlexMatch	$32.44_{\pm 1.99}$	$23.85_{\pm 0.23}$	$19.92_{\pm 0.06}$

Table 10: Benchmark results on STL-10.

Algorithms	Error Rate (40 labels)	Error Rate (250 labels)	Error Rate (1000 labels)
Π-Model [9]	$73.31_{\pm 0.02}$	$54.92_{\pm 3.83}$	$32.22_{\pm 0.66}$
Pseudo-Labeling [4]	$74.48_{\pm 1.48}$	$55.63_{\pm 5.38}$	$31.80_{\pm 0.29}$
VAT [45]	$74.35_{\pm 0.05}$	$56.46_{\pm 2.39}$	$36.48_{\pm 1.74}$
Mean Teacher [10]	$70.62_{\pm 3.04}$	$56.54_{\pm 3.94}$	$32.12_{\pm 0.12}$
MixMatch [12]	$59.46_{\pm 3.49}$	$34.70_{\pm 0.58}$	$20.23_{\pm 0.36}$
ReMixMatch [13]	$36.36_{\pm 3.74}$	$13.36_{\pm 0.26}$	$8.40_{\pm 1.36}$
UDA [11]	$37.31_{\pm 3.03}$	$12.07_{\pm 1.50}$	$6.65_{\pm 0.25}$
FixMatch [14]	$35.42_{\pm 6.43}$	$10.49_{\pm 1.03}$	$6.20_{\pm 0.20}$
FlexMatch	$10.87_{\pm 1.15}$	$7.71_{\pm 0.14}$	$5.56_{\pm 0.20}$

Table 11: Benchmark results on SVHN.

Algorithms	Error Rate (40 labels)	Error Rate (250 labels)	Error Rate (1000 labels)
Π-Model [9]	$69.81_{\pm 1.97}$	$11.76_{\pm 0.62}$	$7.04_{\pm 0.24}$
Pseudo-Labeling [4]	$60.32_{\pm 2.46}$	$14.66_{\pm 0.30}$	$9.56_{\pm 0.25}$
VAT [45]	$52.20_{\pm 25.5}$	$4.89_{\pm 0.03}$	$4.68_{\pm 0.02}$
Mean Teacher [10]	$43.04_{\pm 2.00}$	$3.44_{\pm 0.04}$	$3.27_{\pm 0.00}$
MixMatch [12]	$29.58_{\pm 3.92}$	$4.47_{\pm 0.19}$	$4.08_{\pm 0.12}$
ReMixMatch [13]	$45.12_{\pm 0.27}$	$6.10_{\pm 0.38}$	$5.86_{\pm 0.18}$
UDA [11]	$4.40_{\pm 2.31}$	$1.88_{\pm 0.01}$	$1.93_{\pm 0.01}$
FixMatch [14]	$4.36_{\pm 2.16}$	$2.00_{\pm 0.04}$	$1.97_{\pm 0.03}$
FlexMatch	$5.36_{\pm 2.38}$	$3.99_{\pm 1.78}$	$2.86_{\pm 0.91}$