# Paper Reproducibility Challenge FreeMatch: Self-adaptive Thresholding for Semi-supervised Learning

Yidong Wang, Hao Chen, Qiang Heng, Wenxin Hou, Yue Fan, Zhen Wu, Jindong Wang, Marios Savvides, Takahiro Shinozaki, Bhiksha Raj, Bernt Schiele, Xing Xie

April 16, 2023

**Abstract**

*Semi-Supervised Learning is one of the emerging fields in deep learning due to its direct correspondence to the real-world deployed systems, as the number of labeled data for custom use-case systems might not be present in abundance and the data might be skewed. Semi-supervised learning is also useful in the scalability of the deployed systems, which can consider the site-specific training strategies with every few labeled data for the particular site. Apart from image classification, SSL is drastically making its impact in other computer vision applications such as object detection, tracking, and segmentation. This project aimed to reproduce the important results claimed by the authors of the paper. Following results and experiments were done by taking into account the paper* ***FreeMatch: Self-adaptive Thresholding for Semi-supervised learning****[4]. Wang.et al[4] proposed two algorithms viz.* ***Self Adaptive Thresholding*** *and* ***Self Adaptive Fairness Regularization*** *helped them to give state-of-the-art results on the standard image classification datasets such as CIFAR10, CIFAR100, SVHN, STL10, and ImageNet. The repository is located at https://github.com/shreejalt/freematch-pytorch*

## 1 Disclaimer

All the theorems, propositions, and the proof are taken from the paper by Wang et.al [4]. I have just reproduced the paper to show the main experiments and the results following the propositions of their work in Semi-Supervised Learning. I would like to thank the authors of [4] for their outstanding work on a new approach to semi-supervised learning and detailed analysis of the working of the same. To get into the details of all the loss functions and their proofs, read the original paper FreeMatch: Self-adaptive Thresholding for Semi-Supervised Learning

## 2 Introduction

Recently, there has been extensive research going into semi-supervised image classification that includes different types of pseudo-labeling and consistency regularization methods to use unlabeled data efficiently during the training of CNNs. But most papers like FixMatch[3], ReMixMatch[2], UDA[5](Unsupervised Domain Augmentation) use fixed high thresholds for pseudo-labeling tasks. This makes the training unstable as different classes might need different thresholds based on the hardness and the distribution of the particular class in the dataset and neglects many training examples at the beginning of the training.

To counter this argument, methods like AdaMatch[1] use a warm-up strategy to increase the global threshold so that most unlabelled data is used during the early phase of the training. Also, FlexMatch[6] uses the local threshold learning for each class adaptively based on the confidence of the network output. Although these methods reached very good accuracy numbers on the standard datasets, they did not work when the class distribution is highly skewed or the labeled data is rare. This makes us wonder about adjusting both the local threshold and global threshold simultaneously in the learning process so that the maximum number of unlabeled data is utilized and, at the same time, the skewness constraint, as well as the hardness(in terms of learning) of class, is maintained in the loss propagation.

Wang et.al [4] came up with an interesting approach of **Self Adaptive Thresholding(SAT)** to adaptively learn local-global thresholds of class and dataset, respectively. This helps the authors to differentiate the intra and inter-class discrepancies. They also proposed a **Self Adaptive Class Fairness Regularization(SAF)** function to handle barely supervised settings. It eventually normalizes the batch's distribution with the network's marginal class distribution so that the network does not get biased towards the more recurring class during the training. Two main contributions viz. SAT and SAF regularization are explained below

### 2.1 Formal Definition of Semi-supervised Learning - Modified from [4]

Let $\mathcal{D}_{\mathcal{L}} = \{(x_n, y_n) : n \in [N_L]\}$ and $\mathcal{D}_{\mathcal{U}} = \{(x_n) : n \in [N_U]\}$ be the set of labeled and unlabeled data respectively. Here, $\mathcal{N}_{\mathcal{L}}$ and $\mathcal{N}_{\mathcal{U}}$ are the total number of labeled and unlabeled examples. The supervised loss is given by

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

Here, $p_m(y|\omega(x_b))$ is the probability distribution of the model output and $\omega(x_b)$ is the weak augmentation applied to the input image. $\mathcal{H}$ is the cross-entropy function, $y_b$ is the true label, and $B$ is the batch size of the input.

Consistency regularization or the loss of unlabeled data is defined as

$$\mathcal{L}_u = \frac{1}{\mu B}\sum_{b=1}^{\mu B} \mathbb{1}(max(q_b) > \tau)\mathcal{H}(\hat{q}_b, \mathcal{Q}_b) \tag{2}$$

Here, $\mu$ is the unlabeled ratio, $q_b = p_m(y|\omega(x_b))$, $Q_b = p_m(y|\Omega(x_b))$, $\mathbb{1}(. > \tau)$ acts as an indicator function which masks the loss whose output confidence threshold is greater than $\tau$. $\omega(x_b)$ and $\Omega(x_b)$ are the weak and strong augmentations respectively. Strong augmentations consist of RandAugment, Random Crop, and Horizontal Flipping.

### 2.2 Self Adaptive Thresholding (SAT)[1]

In semi-supervised learning, the threshold $\tau$ plays an important role in selecting the examples for the calculation and stable training. In [4], the authors try to leverage the idea of a learning-aware threshold both at the global and local levels. The adaptive global threshold is learned on the overall prediction of the model's output, whereas the local threshold is molded using the local predictions per class. This implies that the global threshold will be very low in the starting phase of the training and will gradually increase once the training progresses. This will also affect the value of the local threshold functions. The local and global thresholds are learned using the model predictions' Exponential Moving Average(EMA).

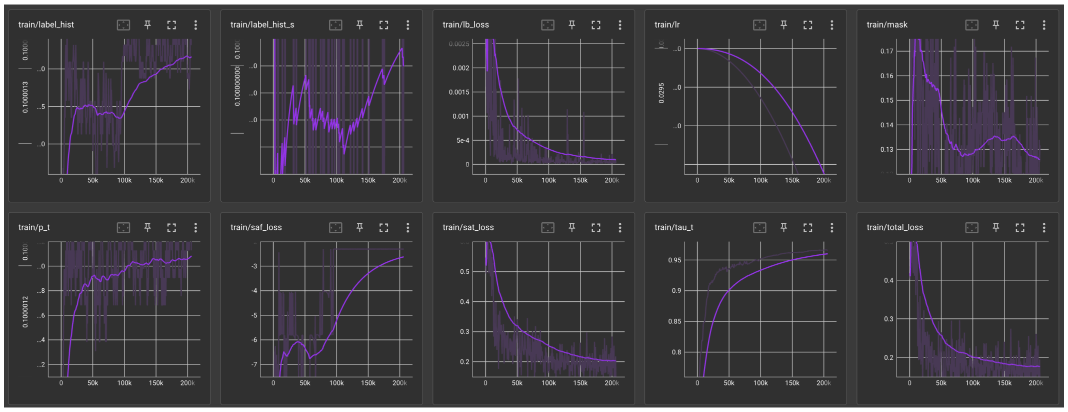

**Figure 1: Logs of CIFAR10 dataset trained using 10 labeled data**

#### 2.2.1 Global Adaptive Threshold

The global threshold is used to modulate the model's confidence in the unlabeled data. Moreover, this threshold will be very less at the start of the training and will gradually increase once the predictions become more confident. As seen in *Fig. 1*, the ***tau_t*** value gradually increases and reaches around 0.98 by 200K iterations. The global threshold is estimated using the EMA of the confidence at each training iteration. Below given is the update equation of the Global SAT(Eq. ref in [4]).

$$\tau_t = \begin{cases} \frac{1}{C}, & t = 0 \\ \lambda\tau_{t-1} + (1-\lambda)\frac{1}{\mu B}\sum_{b=1}^{\mu B} max(q_b), & \text{otherwise} \end{cases} \tag{3}$$

[1]Equations taken from [4]

where $\lambda \in (0,1)$ is used as a momentum decay in EMA and $q_b = p_m(y|\omega(x_b))$. Remember, the calculation of the global threshold is done by detaching the weak logits of the unlabeled data from the computational graph as seen in the *Algorithm 1*

### 2.2.2 Local Adaptive Threshold

The local threshold is a class-centric threshold modulated for each class using the global threshold. Unlike Global SAT, this thresholding technique uses the softmax output of the logits instead of taking the maximum value. The logic behind this technique is to leverage the discrepancies between the classes in terms of the number of labeled data and the hardness of the class(in terms of learning)(Eq. ref in [4]).

$$\tilde{p}_t(c) = \begin{cases} \frac{1}{C}, & t = 0 \\ \lambda \tilde{p}_{t-1}(c) + (1-\lambda)\frac{1}{\mu B}\sum_{b=1}^{\mu B} q_b(c), & \text{otherwise} \end{cases} \tag{4}$$

Here, $\tilde{p}_t(c)$ is the average confidence of the model for the class $c$. Therefore, the final value for the class-wise thresholding is given by

$$\tau_t(c) = \frac{\tilde{p}_t(c)}{max\{\ \tilde{p}_t(c) : c \in [C]\}} * \tau_t \tag{5}$$

At every iteration, the local and global threshold will change. The final consistency loss with the proposed threshold is given by

$$\mathcal{L}_u = \frac{1}{\mu B}\sum_{b=1}^{\mu B} \mathbb{1}(max(q_b) > \tau_t(argmax(q_b)))\mathcal{H}(\hat{q}_b, Q_b) \tag{6}$$

where $\hat{q}_b$ is the pseudo label generated from the weakly augmented logits as shown in ***Algorithm 1***

**Algorithm 1:** PyTorch-like pseudocode for Self Adaptive Threshold (SAT) loss

```
# Inputs:
# logits_ulb_w:  Unlabeled weak augment logits q_b
# logits_ulb_s:  Unlabeled strong augment logits Q_b
# p_t:  EMA of the predictions of the model
# tau_t:  EMA of the global threshold value
# ema_decay:  EMA decay value:  Default is 0.999
with torch.no_grad():
   probs_ulb_w = torch.softmax(logits_ulb_w.detach(), dim=-1)
   # Calculate the probability of the logits
   max_probs_w, max_idx_w = torch.max(probs_ulb_w, dim=-1)
   # Change the global threshold and local predictions using EMA
   tau_t = tau_t * ema_decay + (1.  - ema_decay) * max_probs_w.mean()
   p_t = p_t * ema_decay + (1.  - ema_decay) * probs_ulb_w.mean(dim=0)
   # Generate the mask using the Eq 5
   tau_t_c = (p_t / torch.max(p_t, dim=-1)[0])
   mask = max_probs_w.ge(tau_t * tau_t_c[max_idx_w])
preds_ulb_s = F.log_softmax(logits_ulb_s, dim=-1)
return (F.nll_loss(preds_ulb_s, max_idx_w, reduction = 'none') * mask).mean()
```

## 2.3 Self Adaptive Fairness Regularization (SAF)[1]

The main challenge in a semi-supervised learning setup is when the distribution of the labeled and unlabeled data is skewed and barely supervised (labeled data = 1 per class). The authors tackles this problem by tracking the distribution of the labeled and unlabeled data during the training, which can normalize the skewness of the data during the training. This can help the model not to get biased towards the dominated training class data and help to get over better generalization. This is accomplished by normalizing $_{\mu B}[p_m(y|\Omega(u_b))]$ with the histogram distribution of the pseudo labels.

$$\bar{p} = \frac{1}{\mu B}\sum_{b=1}^{\mu B} \mathbb{1}(max(q_b) > \tau_t(argmax(q_b)))Q_b \tag{7}$$

$$\bar{h} = Hist_{\mu B}(\mathbb{1}(max(q_b) > \tau_t(argmax(q_b))))\hat{Q}_b \tag{8}$$

$$\tilde{h} = \lambda\tau_{t-1} + (1-\lambda)\frac{1}{\mu B}\sum_{b=1}^{\mu B} Hist_{\mu B}(\hat{q}_b) \tag{9}$$

where $\hat{Q}_b$ is the pseudo label generated from the strong augmented image and $\tilde{h}$ is the running average of the histogram calculated from weakly augmented pseudo labels.

The self-adaptive fairness loss is the cross-entropy between the normalized class distributions of strong and weak augmented outputs, as shown in eq 10

$$\mathcal{L}_f = -\mathcal{H}(SumNorm(\frac{\tilde{p}_t}{\tilde{h}_t}), SumNorm(\frac{\bar{p}_t}{\bar{h}_t})) \tag{10}$$

In short, the total loss of training is given by,

$$\mathcal{L} = \mathcal{L}_s + w_u\mathcal{L}_u + w_f\mathcal{L}_f \tag{11}$$

Here, $w_u$ and $w_f$ are the hyperparameters that are tuned for different experiments. The psuedo-code of SAF is shown in Algorithm 2

**Algorithm 2:** PyTorch-like pseudocode for Self Adaptive Fairness (SAF) loss

```
# Inputs:
# logits_ulb_w:  Unlabeled weak augment logits q_b
# logits_ulb_s:  Unlabeled strong augment logits Q_b
# p_t:  EMA of the predictions of the model
# mask:  Mask to consider only high confidence examples calculated in SAT loss
# label_hist:  EMA of the histogram of weak unlabeled data
# ema_decay:  EMA decay value:  Default is 0.999
with torch.no_grad():
   probs_ulb_w = torch.softmax(logits_ulb_w.detach(), dim=-1)
   # Calculate the probability of the logits
   max_probs_w, max_idx_w = torch.max(probs_ulb_w, dim=-1)
   # Get the EMA of histogram distributions
   histogram = torch.bincount(max_idx_w, minlength=p_t.shape[0])
   label_hist = label_hist * ema_decay + (1.  - ema_decay) * (histogram /
    histogram.sum())
logits_ulb_s = logits_ulb_s[mask]
probs_ulb_s = torch.softmax(logits_ulb_s, dim=-1)
max_idx_s = torch.argmax(probs_ulb_s, dim=-1)
# Calculate the histogram of strong logits acc.  to Eq.  8
histogram = torch.bincount(max_idx_s, minlength=logits_ulb_s.shape[1])
histogram /= histogram.sum()
# Calculate SumNorm for the strong and weak normalized distribution ACC to Eq.  10
modulate_p_t = p_t * scaler_p_t
modulate_p_t /= modulate_p_t.sum(dim=-1, keepdim=True)
modulate_prob_s = probs_ulb_s.mean(dim=0, keepdim=True) * scaler_prob_s
modulate_prob_s /= modulate_prob_s.sum(dim=-1, keepdim=True)
loss = (modulate_p_t * torch.log(modulate_prob_s + 1e-9)).sum(dim=1).mean()
return loss
```

## 3 Experiments

This section shows the reproduction results and experiment parameters in detail. The authors did the experiments on five datasets viz. CIFAR10, CIFAR100, SVHN, STL10, and ImageNet. But due to computing constraints, only **CIFAR10** reproduction was possible, which was successful in all the training settings.

### 3.1 Setup

Table 1. shows the experiment setup of the reproducibility challenge. All the hyper parameters and networks used in the experiments are taken from the original paper.

| Specification Type | Name | Value |
|---|---|---|
| CIFAR10 Parameters | Batch Size | 64 |
| | Unlabeled Ratio | 7 |
| | SAT Loss multiplier $w_u$ | 1 |
| | SAF loss multiplier $w_f$ | 0.03 (0.01 for num labeled=10) |
| | Image Size | 32 |
| Network Parameters | WideResNet | Depth: 28, Width: 2 |
| | Mixed Precision Training | ON |
| | Weight Decay | 5e-4 |
| | Learning Rate | 0.03 |
| | SGD Momentum | 0.9 |
| | EMA Decay | 0.999 |

**Table 1: Network and Dataset specifications for the reproduction of experiments**

### 3.2 Results

I successfully reproduced the results mentioned in ***Table 1, Table 7, Table 8, Table 9*** as mentioned in [4]. All the experiments were trained with **Mixed Precision training** on a single Tesla T4 GPU on Google Cloud for around **2^19** iterations (524288) training steps. The original paper trained the model for **2^20** (1048576) iterations.

| Results | # Labels(10) | # Labels(40) | # Labels(250) | # Labels(4000) |
|---|---|---|---|---|
| FreeMatch[4] | 92.93 ± 4.24 | **95.10 ± 0.04** | **95.12 ± 0.18** | **95.9 ± 0.02** |
| **Reproduced** | **93.00** | 94.13 | 95.02 | 95.1 |

**Table 2: Comparision of accuracy results between the original paper and reproduced paper on CIFAR10 dataset with different numbers of labeled data**

As seen in Table 2, the reproduced results are a bit less as all the training was done in mixed precision mode, but the original results posted were done in the normal FP32 training setup. This table refers to the original **Table 1**. in [4]. Also, Table 3. shows the reproduction of the Precision, Recall, F1 Score, and AUC numbers as reported in **Table 8** in [4]

Figure 2 and Figure 3 show the confusion matrix and ROC curves, as mentioned in Figure 3(d) in [4]. All the experiments were run using mixed precision training, so the numbers in Table 3 might differ and may be less by ±0.5 %. All the logs for the experiments conducted with tensorboard logs are available here: Link

## 4 Conclusion

The experiments on the proposed work by the authors were successfully reproduced on the CIFAR10 dataset. The reproduction of the other three datasets can also be accomplished with proper computing resources. All the experiments were done in the AMP environment of PyTorch with half the number of training iterations which might decrease the accuracy numbers as noted in the tables of this paper, but the numbers were very close to the published results of error rates and other metrics.

| Method | Metric Name | Value |
|---|---|---|
| FreeMatch[4] | Precision | 0.8619 |
| | Recall | 0.8593 |
| | F1 Score | 85.23 |
| | AUC | 0.9843 |
| Reproduced | Precision | 0.9312 |
| | Recall | 93.27 |
| | F1 Score | 0.931 |
| | AUC | 0.9957 |

**Table 3: Precision, Recall, F1 Score, and AUC comparison between the methods on CIFAR10(num labeled = 10) . Note that the original paper's authors ran the experiments for different seed values and averaged out the results when reporting. The experiments I did were run only once on a random seed value.**

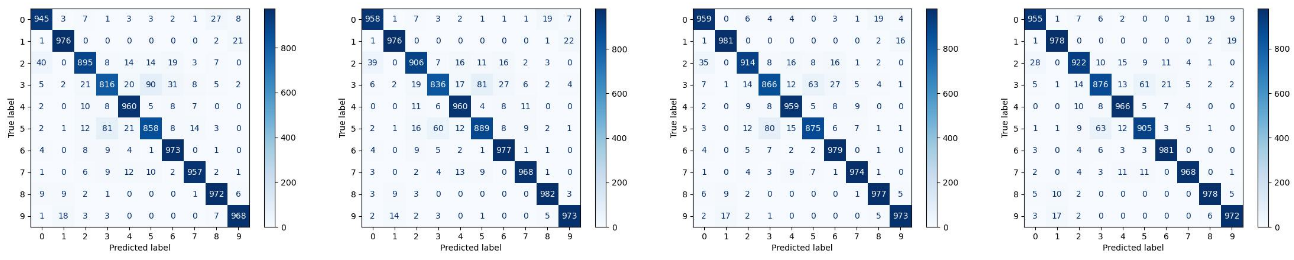

**Figure 2: From Left to Right, Shows the confusion matrix of the CIFAR10 dataset with 10, 40, 250, and 4000 labels, respectively.**

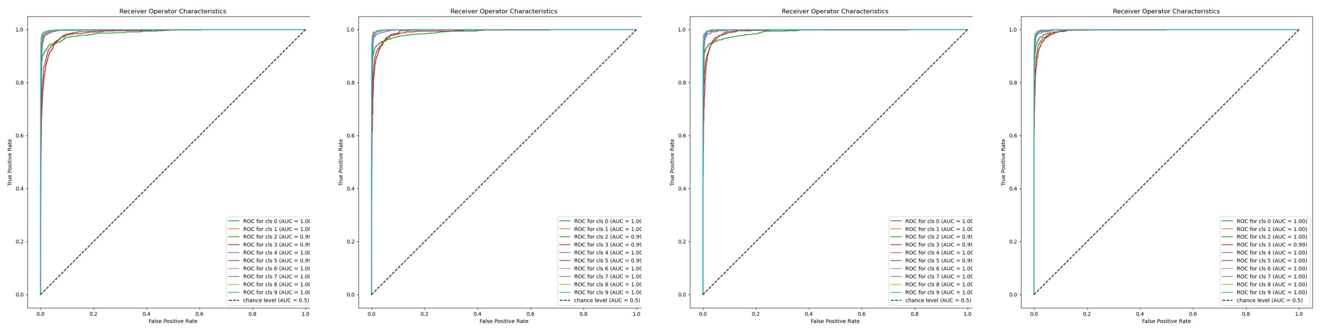

**Figure 3: From Left to Right. Shows the AUC-ROC of the CIFAR10 dataset with 10, 40, 250, and 4000 labels, respectively.**

## References

[1] David Berthelot et al. *AdaMatch: A Unified Approach to Semi-Supervised Learning and Domain Adaptation*. 2022. arXiv: 2106.04732 [cs.LG].

[2] David Berthelot et al. *ReMixMatch: Semi-Supervised Learning with Distribution Alignment and Augmentation Anchoring*. 2020. arXiv: 1911.09785 [cs.LG].

[3] Kihyuk Sohn et al. *FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence*. 2020. arXiv: 2001.07685 [cs.LG].

[4] Yidong Wang et al. *FreeMatch: Self-adaptive Thresholding for Semi-supervised Learning*. 2023. arXiv: 2205.07246 [cs.LG].

[5] Qizhe Xie et al. *Unsupervised Data Augmentation for Consistency Training*. 2020. arXiv: 1904.12848 [cs.LG].

[6] Bowen Zhang et al. *FlexMatch: Boosting Semi-Supervised Learning with Curriculum Pseudo Labeling*. 2022. arXiv: 2110.08263 [cs.LG].