Building One-Shot Semi-supervised (BOSS) Learning up to Fully Supervised Performance

Leslie N. Smith
US Naval Research Laboratory
Washington, DC
leslie.smith@nrl.navy.mil

Adam Conovaloff
NRC Postdoctoral Fellow
US Naval Research Laboratory
Washington, DC
adam.conovaloff.ctr@nrl.navy.mil

Abstract

Reaching the performance of fully supervised learning with unlabeled data and only labeling one sample per class might be ideal for deep learning applications. We demonstrate for the first time the potential for building one-shot semi-supervised (BOSS) learning on Cifar-10 and SVHN up to attain test accuracies that are comparable to fully supervised learning. Our method combines class prototype refining, class balancing, and self-training. A good prototype choice is essential and we propose a practical technique for obtaining iconic examples. In addition, we demonstrate that class balancing methods substantially improve accuracy results in semi-supervised learning to levels that allow self-training to reach the level of fully supervised learning performance. Rigorous empirical evaluations provide evidence that labeling large datasets is not necessary for training deep neural networks. We made our code available at https://github.com/lnsmith54/BOSS to facilitate replication and for use with future real-world applications.

1 Introduction

In recent years deep learning has achieved state-of-the-art performance for computer vision tasks such as image classification. However, a major barrier to the wider-spread adoption of deep neural networks for new applications is that training state-of-the-art deep networks typically requires hundreds or thousands of labeled samples per class to perform at high levels of accuracy and to generalize well.

Unfortunately, manual labeling is labor intensive and might not be practical if labeling the data requires specialized expertise, such as in medical, defense, and scientific applications. In typical real-world scenarios for deep learning, one often has access to large amounts of unlabeled data but lacks the time or expertise to label the required massive numbers needed for training, validation, and testing. An ideal solution might be to achieve performance levels that are equivalent to fully supervised trained networks with only one manually labeled image per class.

In this paper we investigate the potential for building one-shot semi-supervised (BOSS) learning up to achieve comparable performance as fully supervised training. To date, one-shot semi-supervised learning has been little studied and viewed as difficult. We build on the recent observation that one-shot semi-supervised learning is plagued by class imbalance problems [20]. In our context, class imbalance refers to a trained network with near 100% accuracy on a subset of classes and has poor

Preprint. Under review.

performance on other classes. We demonstrate that good prototypes are crucial for successful semi-supervised learning and propose a practical prototype replacement method for the poorly performing classes. Also, we make use of the state-of-the-art in semi-supervised learning methods (i.e., FixMatch [22]) in our experiments. To combat class imbalance, we tested several variations of methods found in the literature for class imbalance problems [12], which refers to the situation where the number of training samples per class vary substantially. We are the first to demonstrate that these methods significantly boost the performance of one-shot semi-supervised learning. Combining these methods with self-training [18] makes it possible on Cifar-10 and SVHN to attain comparable performance as fully supervised trained deep networks.

Unfortunately, we also observed that one-shot semi-supervised learning is more sensitive to hyper-parameters tuning than fully supervised training, which makes training a delicate affair. While this sensitivity can be challenging in practice, we note that this sensitivity can also lead to new opportunities. For example, often researchers propose new network architectures, loss functions, and optimization functions that are tested in the fully supervised regime where small performance gains are used to claim a new state-of-the-art. If these algorithms were instead tested in one-shot semi-supervised learning, more substantial differences in performance would better differentiate methods. Along these lines, we also advocate the use of one-shot semi-supervised learning with AutoML and neural architecture search (NAS) [7] to find optimal hyper-parameters and architectures.

Our contributions are:

1. We rigorously demonstrate for the first time the potential for one-shot semi-supervised learning to reach test accuracies with Cifar-10 and SVHN that are comparable to fully supervised learning.
2. We investigate the value of class balancing for one-shot semi-supervised learning. We introduce four class balancing methods for semi-supervised learning that improve the performance of semi-supervised learning.
3. We propose a practical method for finding iconic prototypes for each class and show that refining a few class prototypes can substantially improve performance.

2 Related Work

Semi-supervised learning: Semi-supervised learning is a hybrid between supervised and unsupervised learning, which combines the benefits of both to better match the scenario of real-world problems. As with supervised learning, semi-supervised learning defines a task (i.e., classification) from labeled data but typically it uses many fewer labeled samples. Like unsupervised learning, semi-supervised learning leverages feature learning from unlabeled data to the greatest extent possible. Semi-supervised learning is a large and mature field and there are several surveys and books on semi-supervised learning methods [33, 25, 5, 32] for the interested reader. In this Section we mention only the most relevant of recent methods.

Recently there have been a series of papers on semi-supervised learning from Google Reseach, including MixMatch [4] , ReMixMatch [3], and FixMatch [22]. MixMatch combines consistency regularization with data augmentation [19], entropy minimization (i.e., sharpening) [9], and mixup [31]. ReMixMatch improved on MixMatch by incorporating distribution alignment and augmentation anchors. Augmentation anchors are similar to pseudo-labeling. FixMatch is the most recent and demonstrated state-of-the-art semi-supervised learning performance. In addition, the FixMatch paper has a discussion on one-shot semi-supervised learning with Cifar-10.

The FixMatch algorithm [22] is primarily a combination of consistency regularization [19, 30] and pseudo-labeling [15]. Consistency regularization utilizes unlabeled data by relying on the assumption that the model should output the same predictions when fed perturbed versions as on the original image. Consistency regularization has recently become a popular technique in unsupervised, self-supervised, and semi-supervised learning [25, 30]. Several researchers have observed that strong data augmentation should not be used when infering pseudo-labels for the unlabeled data but should be employed for consistency regularization [22, 28]. Pseudo-labeling is based on the idea that one can use the model to obtain artificial labels for unlabeled data by retaining pseudo-labels for samples whose probability are above a predefined threshold.

A recent survey of semi-supervised learning [25] provides a taxonomy of classification algorithms. One of the methods in semi-supervised learning is self-training iterations [24, 18] where a classifier is iteratively trained on labeled data plus high confidence pseudo labeled data from previous iterations. In our experiments we found that self-training became reliable once the model's performance is enhanced by prototype refining and class balancing.

Class imbalance: Smith and Conovaloff [20] demonstrated that in one-shot semi-supervised learning there are large variation in class performances, with some classes acheiving near 100% test accuracies while other classes near 0% accuracies. That is, strong classes starve the weak classes, which is analogous to the class imbalance problem [12]. This observation suggests an opportunity to improve the overall performance by actively improving the performance of the weak classes.

We borrowed techniques from the literature on training with imbalanced data [12, 27, 23] (i.e., some classes having many more training samples than other classes) to experiment with several methods for improving the performance of the weak classes. Our experiments demonstrate that these methods substantially improve performance, even when there are the same number of labeled and unlabeled samples for each class. Methods for handling class imbalance can be grouped into two categories: data-level and algorithm-level methods. Data-level techniques [27] reduce the level of imbalance by undersampling the majority classes and oversampling the minority classes. Algorithm-level techniques [23] are commonly implemented with smaller loss factor weights for the training samples belonging to the majority classes and larger weights for the training samples belonging to the minority classes. In our experiments we tested variations of both types of methods and a hybrid of the two.

Class imbalanced semi supervised learning [11] is related to our work but Hyun, et al. addressed the problem space where labeled training data is many more plentiful and the number of both labeled and unlabeled data in each class are vary substantially. Hyun, et al. propose a weighting scheme to under-weight the minority class contribution to the unlabeled loss function, while we instead reduce the weight of the majority classes to the unlabeled loss function, which is more consistent with the class imbalance literature. Li, et al. [16] propose combining self-training with semi-supervised learning for few-shot classification but unlike our method, their method employs a supervised few-shot method for pseudo-labeling.

Meta-learning: Our scenario superficially bears similarity to few-shot meta learning [13, 26, 8, 21], which is a highly active area of research. The majority of the work in this area relies on a large labeled dataset with similar data statistics but this can be an onerous requirement for new applications. While there are some recent efforts in unsupervised pretraining for few-shot meta learning [10, 2], our experiments with these methods demonstrated their inability to adequately perform in one-shot learning to bootstrap our process. Specifically, unsupervised one-shot learning with only five classes obtained a test accuracy of about 50% on high confidence samples and the accuracy dropped sharply when increasing the number of classes.

3 BOSS Methodology

3.1 FixMatch

Since we build on FixMatch [22], we briefly describe the algorithm and adopt the formalism used in the original paper. For an N-class classification problem, let us define $\chi = \{(x_b, y_b) : b \in (1, ..., B)\}$ as a batch of B labeled examples, where x_b are the training examples and y_b are its labels. We also define $\mathcal{U} = u_b : b \in (1, ..., \mu)$ as a batch of μ unlabeled examples where $\mu = r_u B$ and r_u is a hyperparameter that determines the ratio of $\mathcal{U}$ to χ. Let $p_m(y|x)$ be the predicted class distribution produced by the model for input x_b. We denote the cross-entropy between two probability distributions p and q as $H(p, q)$.

The loss function for FixMatch consists two terms: a supervised loss L_s applied to labeled data and an unsupervised loss L_u for the unlabeled data. L_s is the cross-entropy loss on weakly augmented labeled examples:

$$L_s = \frac{1}{B} \sum_{b=1}^{B} H(y_b, p_m(y|\alpha(x_b))) \tag{1}$$

where $\alpha(x_b)$ represent weak data augmentation on sample x_b.

For the unsupervised loss, the algorithm computes the label based on weakly augmented versions of the image as $q_b = p_m(y|\alpha(u_b))$. It is essential that the label is computed on weakly augmented versions of the unlabeled training samples and not on strongly augmented versions. The pseudo-label is computed as $\hat{q_b} = \arg\max(q_b)$ and the unlabeled loss is given as:

$$L_u = \frac{1}{\mu} \sum_{b=1}^{\mu} \mathbb{1}(max(q_b \geq \tau))H(\hat{q_b}, p_m(y|\mathcal{A}(u_b))) \tag{2}$$

where $\mathcal{A}(u_b)$ represents applying strong augmentation to sample u_b and τ is a scalar confidence threshold. The total loss is given by $L = L_s + \lambda_u L_u$ where λ_u is a scalar hyper-parameter. Additional details on the FixMatch algorithm are available in the paper [22].

3.2 Prototype refining

Previous work by Sohn, et al. [22] on one-shot semi-supervised learning relied on the dataset labels to randomly choose an example for each class. The authors demonstrated that the choice of these samples significantly affected the performance of their algorithm. Specifically, they ordered the CIFAR-10 training data by how representative they were of their class by utilizing fully supervised trained models and found that using more prototypical examples achieved a median accuracy of 78% while the use of poorly representative samples failed to converge at all. The authors acknowledged that their method for finding prototypes was not practical. In contrast, we now present a practical approach for choosing an iconic prototype for each class.

In real-world scenarios, one's data is initially all unlabeled but it is not overly burdensome for an expert to manually sift through some of their dataset to find one iconic example of each class. In choosing iconic images of each class, the labeler's goal is to pick images that represent the class objects well, while minimizing the amount of background distractors in the image. In our own experiments with labeled datasets Cifar-10 and SVHN, we did not rely on the labels but reviewed a small fraction of the training data to manually choose class prototypes.

In addition, we also propose a simple iterative technique for improving the choice of prototypes because good prototypes are crucial to good performance. After choosing prototypes, the next step is to make a training run and examine the final class accuracies. For any class with poor accuracies relative to the other classes, it is likely that a better prototype can be chosen. We recommend returning to the unlabeled dataset to find replacement prototypes for only the poorly performing classes. In our experiments we found doing this even once to be beneficial. In addition, our future plans include investigating potential performance improvements by preprocessing prototype images to minimize background distractors.

One might argue that prototype refining is as much work as labeling several examples per class and using many training samples will make it easier to train the model. From only a practical perspective, labeling five or ten examples per class is not substantially more effort relative to labeling only one iconic example per class and prototype refining. While in practice one may want to start with more than one example for ease of training, there are scientific, educational, and algorithmic benefits to studying one-shot semi-supervised learning, which we discuss in our Broader Impact statement.

3.3 Class balancing

We believe a class imbalance problem is an important factor in training neural networks, not only in one-shot semi-supervised learning but also a factor for small to mid-sized datasets. A network with random weights usually outputs a single class label for every sample (i.e., randomly initialized networks do not generate random predictions). Hence, all networks start their training with elements of the class imbalance problem but the presence of large, balanced training data allows the network to overcome this problem. Since class imbalance is always present when training deep networks, class balancing methods might always be valuable, particularly when training on one-shot, few-shot, or small labeled datasets, and we leave further investigations of this for future work.

For class balancing, our algorithm first computes the number of pseudo-labels generated in each class and uses this measure as a surrogate for model's class imbalance. Specifically, as the algorithm computes the pseudo-labels for all of the unlabeled training samples, it counts the number that fall within each class, which we designate as $\mathcal{C} = c_n : n \in (1, ..., N)$ where N is the number of classes.

We assume a similar number of unlabeled samples in each class so the number of pseudo-labels in each class should also be similar.

Our first class balancing method is based on oversampling minority classes. Our algorithm reduces the pseudo-labeling thresholds for minority classes to include more examples of the minority classes in the training. Formally, in pseudo-labeling the following unsupervised loss function is used for the unlabeled data in place of Equation 2:

$$L_u = \frac{1}{\mu} \sum_{b=1}^{\mu} \mathbb{1}(max(q_b \geq \tau_n)) H(\hat{q_b}, q_b) \tag{3}$$

where $q_b = p_m(y|\mathcal{A}(u_b))$, $\hat{q_b} = \arg\max(q_b)$, and τ_n is the class dependent threshold for inclusion in the unlabeled loss L_u. We define the class dependent thresholds as:

$$\tau_n = \tau - \Delta(1 - \frac{c_n}{max(\mathcal{C})}) \tag{4}$$

where c_n is the number of pseudo-labeled in class n and Δ is a scalar hyper-parameter ($\tau > \Delta > 0$) guiding how much to lower the threshold for minority classes. Hence, the most frequent class will use a threshold of τ while minority classes will use lower thresholds, down to $\tau - \Delta$.

The next two class balancing methods are variations on loss function class weightings. In the FixMatch algorithm, all unlabeled samples above the threshold are included in Equation 3 with the same weight. Instead, our second class balancing algorithm becomes:

$$L_u = \frac{1}{Z\mu} \sum_{b=1}^{\mu} \mathbb{1}(max(q_b \geq \tau_n)) H(\hat{q_b}, q_b)/c_n \tag{5}$$

where the loss terms are divided by c_n and Z is a normalizing factor that makes L_u the same magnitude as without this weighting scheme (this allows the unlabeled loss weighting λ_u to remain the same).

Our third class balancing algorithm is identical to the previous method except it uses an alternate class count $\hat{c}_u$ in Equation 5. We define $\hat{c}_u$ using only the high confidence pseudo-labeled samples above the threshold. The intuition of this third method is that each of the classes should contribute equally to the loss L_u (i.e., each sample's loss is divided by the number of samples of that class included in L_u). In practice, this method's weights might be an order of magnitude larger than the previous method's weights, which might contribute to training instability, so we compare both methods in Section 4.2.

3.4 Self-training iterations

Labeled and unlabeled data play different roles in semi-supervised learning. Here we propose self-training iterations where the pseudo-labels of the highest confidence unlabeled training samples are combined with labeled samples in a new iteration. Increasing the number of labeled samples per class improves performance, and substantially reduces training instability and performance variability. Although some of these pseudo-labels might be wrong, we rely on the observation that the training of deep networks are robust to small amounts of labeling noise (i.e., labeling noise of less than 10% does not harm the trained network's performance [1]). Hence, we aimed to achieve a 90% accuracy from semi-supervised learning with the class balancing methods.

Self-training is done in BOSS by adding to the testing stage a computation of the model predictions on all of the unlabeled training data. These are sorted from the the highest prediction probabilities down and the dataset is saved. After the original training run, the labeled data can be combined with a number of the highest prediction samples from each class and a subsequent self-training iteration run can use the larger labeled dataset for retraining a new network. We experimented with labeling 5, 10, 20, and 40 of the top predictions per class and the results are reported in Section 4.3.

set	airplane	auto	bird	cat	deer	dog	frog	horse	ship	truck	Total
1	29	98	71	89	97	16	98	97	97	97	79
2	28	99	70	43	97	89	98	97	98	0	72
3	96	98	63	20	97	96	98	87	98	97	86
4	29	98	65	10	96	32	98	97	97	96	72
5	28	97	70	46	96	48	53	76	96	97	72
6	80	98	71	52	97	92	98	87	98	97	82
7	28	99	75	54	95	86	95	86	96	94	83

Table 1: One-shot semi-supervised average (of 2 runs) class accuracies for Cifar-10 test data with the FixMatch model, that was trained on sets of manually chosen prototypes for each class. Prototype set 6 was modified from set 2 and prototype set 7 was modified from set 4 (i.e., prototype refining).

4 Experiments

In this Section we demonstrate that the BOSS algorithms can achieve comparable performance with fully-supervised training of Cifar-10 [14] and SVHN [17]. We compare our results to FixMatch[1] [22] and demonstrate the value of our approach. Our experiments use a Wide ResNet-28-2 [29] that matches the FixMatch reported results and we used the same cosine learning rate schedule described by Sohn, et al. [22]. Our hyper-parameters were in a small range and the specifics are provided in the Appendix. For data and data augmentation, we used the default augmentation in FixMatch but our experiments did show a small improvement in using RandAugment [6] for strong data augmentation. Our runs with fully supervised learning of the Wide ResNet-28-2 model produced a test accuracy of $94.9 \pm 0.3\%$ for Cifar-10 [14] and test accuracy of $98.26 \pm 0.04\%$ for SVHN [17], which we use for our basis of comparison. We made our code available at `https://github.com/lnsmith54/BOSS` to facilitate replication and for use with future real-world applications.

4.1 Choosing prototypes and prototype refining

For our experiments with Cifar-10, we manually reviewed the first few hundred images and choose five sets of prototypes that we will refer to as class prototype sets 1 to 5. However, the practioner need only create one set of class prototypes and can perform prototype refining, as we describe below.

Table 1 presents the averaged (over two runs) test accuracies for each class, computed from FixMatch on the Cifar-10 test dataset for each of the prototype sets 1 to 5. This Table illustrates that a good choice of prototypes (i.e., set=3) can lead to good performance in all the classes. Table 1 also shows that for other sets the class accuracies can be quite high for some classes while low for other classes. Hence, the poor performance of some classes implies that the choice of prototypes for these classes in those sets can be improved. In prototype refining, one simply reviews the class accuracies to find which prototypes should be replaced.

We demonstrate prototype refining with two examples. The airplane and truck class accuracies in set 2 are poor so we replaced these two prototypes and name this set 6. In set 4, the cat and dog classes are performing poorly so we replaced these two prototypes and name this set 7. Table 1 shows the class accuracies for sets 6 and 7 and these results are better than the original sets. More importantly, the balancing of the accuracies across all the classes enables the use of this trained model to automatically generate labeled examples for self-training, as described in Section 3.4.

4.2 Class balancing

In this Section we report the results from FixMatch and demonstrate substantial improvements with the class balancing methods in BOSS. Table 2 presents our main results, which illustrates the benefits from prototype refining, class balancing, and one self-training iteration. The rows in the table list the results for five sets of class prototypes (i.e., 1 prototype per class) for Cifar-10. Rows for sets 6 and 7 provides the results for prototype refining of the original sets 2 and 4, respectively. The

[1] With appreciation, we acknowledge the use of the code kindly provided by the authors at `https://github.com/google-research/fixmatch`

		BOSS balance method				Self-training			
set	FixMatch	1	2	3	4	+5	+10	+20	+40
1	79 ± 1 [61]	**91.4 ± 2** [75]	90 ± 5 [85]	84 ± 6 [70]	88 ± 2 [67]	94.8 ±0.1	95.2 ±0.1	95.2 ±0.1	95.2 ±0.1
2	74 ± 5 [58]	**91.8 ± 1** [85]	90 ± 3 [83]	88 ± 2 [81]	80 ± 14 [89]	93.6 ±0.2	95.1 ±0.1	95.1 ±0.3	95.1 ±0.2
3	86 ± 1 [88]	92.8 ± .2 [93]	91 ± 2 [91]	91 ± 3 [89]	**92.8 ± .1** [87]	94.6 ±0.5	94.8 ±0.5	94.9 ±0.1	95.2 ±0.1
4	74 ± 8 [73]	77.7 ± .3 [89]	81 ± 6 [72]	81 ± 8 [86]	**90 ± 7** [82]	94.9 ±0.1	94.9 ±0.4	94.9 ±0.5	95.1 ±0.3
5	69 ± 7 [69]	86 ± 7 [86]	89 ± 6 [73]	83 ± 10 [87]	**90 ± 3** [85]	89.6 ±0.3	95.2 ±0.1	95.2 ±0.2	95.2 ±0.1
6	82 ± 0.6 [87]	91.5 ± 1 [83]	92 ± .7 [81]	91.8 ± 1 [75]	**92 ± 1** [70]	94.6 ±0.1	95.1 ±0.2	94.7 ±0.1	94.9 ±0.1
7	78 ± 0.1 [56]	91.7 ± .3 [68]	92.3 ± .8 [79]	91.1 ± 2.5 [62]	**93 ± .3** [66]	94.9 ±0.1	94.7 ±0.2	94.9 ±0.1	95.1 ±0.1

Table 2: Main results. BOSS methods are compared using five sets of class prototypes (i.e., 1 prototype per class) for Cifar-10, plus two sets from prototype refining. The FixMatch column shows test accuracies (average and standard deviation of 4 runs) for the original FixMatch code on the prototype sets. The next four columns gives the accuracy results for the class balance methods (see text for a description of class balance methods). Results for the PyTorch reimplementation of FixMatch and modified with the BOSS methods are shown in brackets [.]. The self-training iteration was performed with the top pseudo-labels from the run shown in bold and the results are in the next four columns.

FixMatch column shows results (i.e., average and standard deviation over four runs) for the original FixMatch code on the prototype sets. The number within brackets [.] are results from a PyTorch reimplementation of FixMatch, that we discuss below.

The next four columns presents the BOSS results with class balancing methods. As described in Section 3.3, class balance method 1 represents oversampling of minority classes, balance methods 2 and 3 are two forms of class-based loss weightings, and balance = 4 is a hybrid that combines balance methods 1 and 3. The use of class balancing significantly improves on the original FixMatch results, with increases of up to 20 absolute percentage points. Generally, the hybrid class balance method 4 is best, except when instabilities hurt the performance. Crucially, the performance is generally in the 90% range with good performance across all the classes, which enables the self-training iteration.

Table 2 indicates that good class prototypes (i.e., sets 3, 6, and 7) result in test accuracies near 90% and low variance between runs. However, when some of the class prototypes are inferior, some of the of the training runs exhibit instabilities that cause lower averaged accuracies and higher variance. Other experiments in our Supplemental Materials demonstrate that in these cases, reducing the amount of class balancing reduces the instabilities (i.e., the quality of the class prototypes governs the hyper-parameter values).

PyTorch version: We have taken advantage of a PyTorch reimplementation[2] of the original TensorFlow version of the FixMatch code to test our proposed BOSS methods in PyTorch. Table 2 reports the best test accuracies for the PyTorch version in the brackets [.].

It is clear to us that the researcher who reimplemented FixMatch in PyTorch took care to replicate FixMatch. In training with 4 labeled samples per class, his code obtained a test accuracy of 89 ± 5% for Cifar-10, compared to results of 87 ± 3% reported in the paper. However, it is also clear from our experiments and Table 2 that there are substantial differences between the TensorFlow and PyTorch versions when comparing one-shot semi-supervised learning. A possible source of this difference might come from the preprocessing step in the TensorFlow implementation. This preprocessing includes a sorting process of the unlabeled data that is not present in the PyTorch code. This preprocessing was not mentioned in Sohn, et al. and could easily be deemed inconsequential but it

[2] With appreciation, we acknowledge the use of the code provided at `https://github.com/CoinCheung/fixmatch`

		BOSS balance method				self-training			
set	FixMatch	1	2	3	4	+5	+10	+20	+40
1	95.9 ± 3	$\mathbf{97.4 \pm .2}$	$96.4 \pm .9$	95.7 ± 1.6	$96.8 \pm .1$	97.9	97.9	97.9	97.8
2	91.5 ± 3	$\mathbf{97.4 \pm .1}$	$97.1 \pm .1$	$97.1 \pm .1$	$95.6 \pm .1$	94.1	97.9	97.6	97.7
3	$93.9 \pm .1$	$\mathbf{97.3 \pm .3}$	$97.2 \pm .2$	92 ± 7	$91.3 \pm .3$	97.8	97.9	97.8	97.9
4	89.2 ± 12	$\mathbf{96.5 \pm .6}$	90 ± 10	89 ± 11	83 ± 16	97.6	96.7	97.0	98.0

Table 3: BOSS methods are compared using four sets of class prototypes (i.e., 1 prototype per class) for SVHN. The FixMatch column shows results for the original FixMatch code on the prototype sets. The next four columns gives the accuracy results for the class balance methods Results are an average of test accuracies for four runs. The self-training iteration was performed on the results from the class balancing shown in bold.

does seem to impact the trained network's performance on one-shot semi-supervised learning. Hence, we observe that the sensitivity of one-shot semi-supervised learning reveals even minor differences that are invisible in fully supervised learning.

The PyTorch implementation also shows that the class balancing methods improve the test accuracy over FixMatch. In particular, class balance method 1 (i.e., oversampling) appears to improve the test accuracy more than the other methods.

4.3 Self-training iterations

The final four columns of Table 2 list the results of performing one self-training iteration. The self-training was initialized with the original single labeled sample per class, plus the most confident pseudo-labeled examples from the BOSS training run that is highlighted in bold. For example, the '+5' columns means that five pseudo-labeled examples per class were combined with the original labeled prototypes to make a set with a total of 60 labeled examples. These self-training results demonstrate that one-shot semi-supervised learning can reach comparable performance to the results from fully supervised training (i.e., 94.9%), often with adding as few as 5 samples per class. However, we expect that in practice, self-training by adding more samples per class will prove more reliable.

4.4 SVHN

SVHN is obtained from house numbers in Google Street View images and is used for recognizing digits (i.e., 0 – 9) in natural scene images. Visual review of the images show that the training samples are of poor quality (i.e., blurry) and often contain distractors (i.e., multiple digits in an image). Because of the quality issue, we needed to review several hundred unlabled training samples in order to find four class prototype sets.

Even though the SVHN training images are of poorer quality than the Cifar-10 training images, one-shot semi-supervised learning with FixMatch on sets of prototypes produced higher test accuracies than with Cifar-10. Table 3 presents equivalent results for the SVHN dataset as reported in Table 2 for Cifar-10. Since the results for FixMatch are all above 89%, we did not perform prototype refining on any of these sets. However, here too the class balancing methods increase the test accuracies above the FixMatch results. With these four class prototype sets, class balance method 1 produces the best results. The test accuracies from balance method 1 are approximately 1% lower than the fully supervised results of $98.26 \pm 0.04\%$. The improvements from self-training were small and the best results fell about 0.5% below the results of of fully supervised training. We believe the differences between Cifar-10 and SVHN are related to the natures of the datasets.

5 Conclusions

The BOSS methodology relies on simple concepts: choosing iconic training samples with minimal background distractors, employing class balancing techniques, and self-training with the highest confidence pseudo-labeled samples. Our experiments in Section 4 demonstrate the potential of training a network with only one sample per class and have confirmed the importance of class balancing methods. BOSS bring one-shot and few-shot semi-supervised learning closer to reality for applications with large amounts of unlabeled data.

Our work provides researchers with the following observations and insights:

1. There is evidence that labeling a large number of samples might not be required for training deep neural networks.
2. All networks have a class imbalance problem to some degree. Examining class accuracies relative to each other provides insights into the network's training.
3. Each training sample can affect the training. One-shot semi-supervised learning provides a mechanism to study the atomic impact of a single sample. This opens up the opportunity to investigate the factors in a sample that help or hurt training performance.
4. The PyTorch reimplementation of FixMatch showed substantial differences from the TensorFlow version that were not apparent when training with four or more samples per class. This sensitivity of one-shot semi-supervised learning can be used with AutoML and Neural Architecture Search (NAS) to obtain optimal hyper-parameters and models. In addition, we recommend that researchers test their novel architectures, loss and optimization functions on one-shot semi-supervised learning to better differentiate their methods.

Broader Impact

It is widely accepted that large labeled datasets are an essential component of training deep neural networks, either directly for training or indirectly via transfer learning. To the best of our knowledge, this paper is the first to demonstrate performance comparable to fully supervised learning with one-shot semi-supervised learning. Eliminating the burden of labeling massive amounts of training data creates great potential for new neural network applications that attain high performance, which is especially important when labeling requires expertise. Hence, the societal impact will be to make deep learning applications even more widespread.

From a scientific perspective, one-shot semi-supervised learning provides important insights on the intricacies of training deep neural networks. The effect of changing just one training image can significantly impact the final performance. Unlike fully supervised learning that commonly deals with the training of large datasets, this method provides a technique to gain information about the impact of a single labeled sample in training. In addition, we anticipate that further investigation into the instability issues of one-shot semi-supervised learning will lead to new understandings of training neural network.

Furthermore, the experience of training highly sensitive networks provides an educational experience on hyper-parameter tuning that carries over to easier training situations. In order to achieve convergence with one-shot semi-supervised learning, one must learn how to tune the hyper-parameters and architecture well. Similarly, we believe that utilizing one-shot semi-supervised learning with automatic methods such as AutoML and neural architecture search (NAS) will lead to better choices for hyper-parameters and architectures.

Limitations: While our work has taken valuable steps towards making one or few-shot semi-supervised learning possible for applications, a large gap still remains before this can be realized in practice, especially due to issues with stability during training and hyper-parameter sensitivity. The sensitivity of the results to choices of the hyper-parameters makes one-shot semi-supervised learning difficult to use in real-world applications. While there is a wide range of valuable applications (e.g., medical) that could benefit from semi-supervised learning, the testing of these applications is beyond the scope of this work.

While we attempted to provide a thorough investigation, there are a number of limitations in our work and several factors that we did not have sufficient time to explore. Our implementation was built on state-of-the-art FixMatch algorithm but the ideas presented here should carry over to other semi-supervised learning methods (this was not tested in our experiments). The model used in our experiments was a Wide ResNet-28-2; other architectures were not compared.

In addition, we made use of labeled test data to demonstrate the performance of BOSS. In practical settings, one has a large unlabeled dataset and one wishes to avoid burdensome manual labeling. However, the samples in the test dataset are less important than the choices for the class prototypes, so a small test dataset can be quickly created from the "discards" when searching for iconic prototypes. A small test dataset is useful for prototype refinement (i.e., deciding which class prototypes to replace) and it provides the practitioner with useful feedback on the system's performance with a little additional effort. But even without any test data, one can utilize the pseudo-labeled class counts to decide which class prototypes should be replaced.

Furthermore, there are several assumptions that might not hold true in a practical setting. First of all, there is an implicit assumption that the unlabeled dataset is class balanced; that is, it contains the same number of samples of each class. In practical situations with large amounts of unlabeled data, this assumption is unlikely to be true. In cases where the number of unlabeled samples belonging to each class can be estimated, it is possible to adapt the class balancing methods. When the number of unlabeled samples belonging to each class is unknown, it is possible to create a small validation set in a similar manner as described above for creating a test set and utilize the validation set as a measure of class balance.

In addition, we also assume in our experiments that all of the unlabeled samples belong to one of the known classes. In practical settings, the unlabeled dataset might contain samples that don't belong to any of the prototype classes. We did not test the situation where we use only a subset of the classes in the training datasets.

Acknowledgments and Disclosure of Funding

With appreciation, we acknowledge the use of the code kindly provided by the authors of FixMatch [22] at `https://github.com/google-research/fixmatch` and the author of a PyTorch reimplementation where the code was provided at `https://github.com/CoinCheung/fixmatch`. We also express our deep appreciation to Nicholas Carlini and David Berthelot for discussions of ReMixMatch and FixMatch that facilitated our use of their methods and codes.

This work was funded by the Office of Naval Research. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the US Navy.

References

[1] Görkem Algan and Ilkay Ulusoy. Image classification with deep learning in the presence of noisy labels: A survey. *arXiv preprint arXiv:1912.05170*, 2019.

[2] Antreas Antoniou and Amos Storkey. Assume, augment and learn: Unsupervised few-shot meta-learning via random labels and data augmentation. *arXiv preprint arXiv:1902.09884*, 2019.

[3] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019.

[4] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 5050–5060, 2019.

[5] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542, 2009.

[6] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2019.

[7] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[9] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005.

[10] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

[11] Minsung Hyun, Jisoo Jeong, and Nojun Kwak. Class-imbalanced semi-supervised learning. *arXiv preprint arXiv:2002.06815*, 2020.

[12] Justin M Johnson and Taghi M Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

[13] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

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

[15] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 2, 2013.

[16] Xinzhe Li, Qianru Sun, Yaoyao Liu, Qin Zhou, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to self-train for semi-supervised few-shot classification. In *Advances in Neural Information Processing Systems*, pages 10276–10286, 2019.

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

[18] Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. Semi-supervised self-training of object detection models. *WACV/MOTION*, 2, 2005.

[19] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in neural information processing systems*, pages 1163–1171, 2016.

[20] Leslie N Smith and Adam Conovaloff. Empirical perspectives on one-shot semi-supervised learning. *arXiv preprint arXiv:2004.04141*, 2020.

[21] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[22] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020.

[23] Yanmin Sun, Mohamed S Kamel, Andrew KC Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[24] Isaac Triguero, Salvador García, and Francisco Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information systems*, 42(2):245–284, 2015.

[25] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[26] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[27] Shuo Wang and Xin Yao. Multiclass imbalance problems: Analysis and potential solutions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(4):1119–1130, 2012.

[28] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.

[29] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[30] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE international conference on computer vision*, pages 1476–1485, 2019.

[31] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[32] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

[33] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

A Appendix

A.1 Hyper-parameters

For FixMatch we used the default hyper-parameters that were specified in Sohn, et al. [22]. However, in our initial experiments with the class balance methods, we found that these hyper-parameters performed poorly. Therefore, we used a different set of hyper-parameter values for FixMatch and for the BOSS methods.

Table 4 contains the hyper-parameter values used for the results reported in our paper. Additional hyper-parameter settings that were consistent over all the runs include setting kimgs = 32768 (i.e., the number of training images) and $\lambda_u = 1$ (i.e., the unlabeled loss multiplicative factor). Furthermore, we set the augment input parameter to 'd.d.d', which is the default data augmentation for the labeled and unlabeled data. Our early experiments with setting the augment input parameter to 'd.d.rac' produces small improvements so we subsequently used the default values. The balance column reflects the class balancing method used (balance = 0 corresponds to FixMatch, which does not use any class balancing method). The remaining columns specify the weight decay, learning rate, batch size, momentum, ratio of unlabeled to labeled data, confidence threshold, and change in the confidence threshold for minority classes. Details of these last three hyper-parameters are provided in the main text.

Specifically, we found that increasing the ratio of unlabeled to labeled data (from 7 to 9), weight decay (from 5×10^{-4} to 8×10^{-4}) and the learning rates (from 0.03 to 0.06) improved performance. We also found that decreasing the confidence threshold from 0.95 to 0.9 improved performance but for class balancing methods 1 and 4, we left the confidence threshold at 0.95 because the class-based thresholds were lowered by these class balancing methods. We also discovered that a smaller batch-size improved performance and chose a batch size of 30 that was a multiple of the number of classes. Our experiments with momentum found a small improvement with values between 0.85 and 0.9 and settled on using 0.88 for our experiments.

As mentioned above, we tried to use the same hyper-parameters for both FixMatch and for the class balancing methods but this proved to provide an unfair comparison to one or the other. Table 5 illustrates this. This Table provides the averaged test accuracies for class prototype set 2 for the default and another choice of weight decay (WD), learning rate (LR), batch size (BS), and the ratio of the unlabeled to labeled data (r_u). The results for the BOSS methods improve significantly by tuning the hyper-parameters but the performance of FixMatch is reduced substantially. So we used the default set of hyper-parameters for FixMatch and another set of hyper-parameter values for the class balance methods.

Method	balance	weight decay	LR	Batch	Momentum	r_u	τ	Δ
FixMatch	0	5×10^{-4}	0.03	64	0.88	7	0.95	0
Cifar training	1, 4	8×10^{-4}	0.06	30	0.88	9	0.95	0.25
Cifar training	2, 3	8×10^{-4}	0.06	30	0.88	9	0.9	0
Self-training	4	5×10^{-4}	0.03	64	0.88	7	0.95	0.25
SVHN training	1, 4	6×10^{-4}	0.04	32	0.85	7	0.95	0.25
SVHN training	2, 3	6×10^{-4}	0.04	32	0.85	7	0.9	0
Self-training	0	6×10^{-4}	0.04	32	0.85	7	0.95	0.25

Table 4: Hyper-parameter values for each of the various steps in the training.

		BOSS balance method			
WD/LR/BS/r_u	FixMatch	1	2	3	4
$5 \times 10^{-4}/0.03/64/7$	74 ± 5	34 ± 2	44 ± 7	40 ± 2	31.5 ± 0.5
$8 \times 10^{-4}/0.06/30/9$	47 ± 8	93 ± 0.7	90 ± 2	84 ± 13	78 ± 20

Table 5: Test accuracies for class prototype set 2 for two hyper-parameter settings. The hyper-parameters are weight decay (WD), learning rate (LR), batch size (BS), and the ratio of the unlabeled to labeled data (r_u).

A.2 Implementation details

In this Section we describe the changes we made to the original FixMatch codes and provide guidance on how to replicate our experiments. This Section relies on the reader being familiar with the TensorFlow version at `https://github.com/google-research/fixmatch` and the PyTorch version located at `https://github.com/CoinCheung/fixmatch`. We provide a copy of our codes as part of our Supplemental Materials.

Modifications to the original TensorFlow version of the FixMatch code were localized. In the TensorFlow version, the primary changes were made to *fixmatch.py*. This includes the implementation of the four class balancing methods. In support of these methods, the code for computing the number of pseudo-labels in each class was implemented. Also, a few new input parameters were added to this file that are related to the class balancing methods. Specifically, we added the input parameter "balance" to specify the class balancing method (balance=0 acts the same as the original FixMatch code) and "delT" (i.e., Δ) as the amount that balance method 1 can reduce the threshold. Modifications were also made to *cta/lib/train.py* to compute test accuracies for each class, keep track of the best test accuracy, and output the sorted pseudo-labels for the unlabeled training data. In addition, changes to *libml/data.py* and *libml/augment.py* were required in order to accept the new prototype versions of the labeled datasets.

In addition to the code, the TensorFlow FixMatch version required several other steps that are supported by code in the *scripts* folder. Instructions for creating the necessary dataset files are located on the website at `https://github.com/google-research/fixmatch`. These instructions use programs in the scripts folder that needed to be modified in order to create the dataset files needed for the prototype sets and for self-training.

We named the prototype datasets with a 'p' at the end to distinguish them from the original datasets. That is, 'cifar10' became 'cifar10p' and 'svhn' became 'svhnp'. Therefore, it was necessary to create *scripts/cifar10_prototypes.py* and *scripts/svhn_prototypes.py* to generate the labeled training data files. We note that to be consistent with the TensorFlow FixMatch, we used 'seed' as the input parameter to represent different prototype sets. It is also necessary to copy the unlabeled training and labeled training files from the cifar10/svhn file names to the cifar10p/svhnp file names and we provide shell scripts to do so.

Self-training is performed as a separate step from the first training run. The training run will have created three files containing the pseudo-labels for the unlabeled training data sorted from the most confident predictions down. The three files are the pseudo-labels, the confidences, and the true labels (used only for debug purposes). The programs *scripts/cifar10_iteration.py* and *scripts/svhn_iteration.py* are provided to combine the highest confidence pseudo-labeled examples with the labeled class prototypes and create the necessary files for the self-training run. We provide shell scripts as a template for how this is done. Once these files are created, the self-training iteration can be run.

Most of our experiments were run on a SuperMicro SuperServer with Tesla V100 GPUs. We discovered that it was important to run our experiments on only 1 GPU and all our runs using multiple GPUs performed poorly.

Modifications to the PyTorch version of the FixMatch code were simpler than for the TensorFlow code. However, the execution of this code ran almost three times longer, which greatly reduced the number of experiments we could run due to constraints on computational resources. The primary modifications for class balancing were added to *label_guessor.py*. Secondary modification were made to the main program in *train.py* to add the class balancing input parameters and arguments for the call to label_guessor. In addition, *cifar.py* was modified to use the class prototypes instead of random

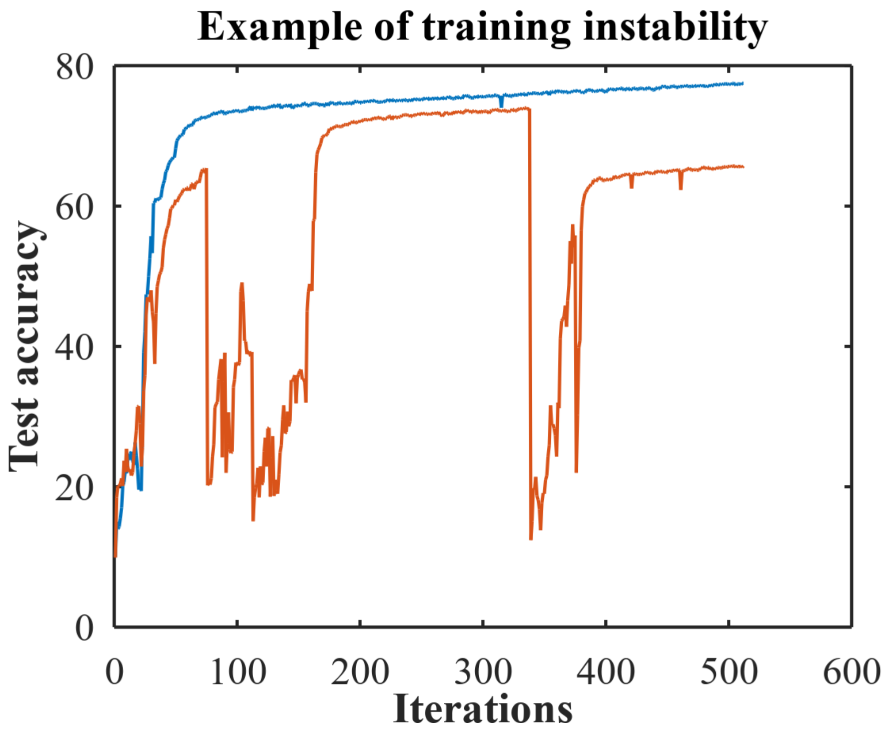

Figure 1: An example of training to a poor local minimum (blue) and training with instabilities (red). Both end with poor test accuracies but for different reasons.

examples. It was not necessary to create class prototype files as it was with the TensorFlow version. We did not have sufficient time to test self-training with the PyTorch version.

A.3 Discussion of training instabilities, poor local minimum, and hyper-parameter sensitivity

In our experiments we observed sensitivity of one-shot semi-supervised learning performance to the choices for the hyper-parameters and the class prototypes sets. That is, we observed that good choices for the prototypes and prototype refining significantly reduced the instabilities and the variability of the results (i.e., few instabilities were encountered for Cifar-10 prototype sets 3, 6, and 7 so the final accuracies were higher and the standard deviations of the results were lower). In sets where the performance was inferior, there was always at least one class that performed poorly. However, we also found that the hyper-parameter values made a significant difference.

We investigated the cases of poor performance and discovered that there were two different situations. Figure 1 provides examples of test accuracies during the training for both situations. The blue curve is the test accuracy where in one training run the network learns a final test accuracy of 77% for the case of class prototype set 4, balance method 3 and the hyper-parameters correspond to those described in Section A.1 (on the other hand, another run with the same hyper-parameters produced an accuracy of 93%). We hypothesize that in this situation the network can get stuck in a poor local minimum. The red curve in Figure 1 is an example of the other case and here the test accuracy in one training run learns a final test accuracy of 65% (i.e., for class prototype set 5 and balance method 3). Clearly the behavior during training is different in this case because the training is dominated by instabilities (i.e., where the model suddenly diverges during training).

We found that these two situations are the two sides of problem and it is important when tuning the hyper-parameters to identify which one is occurring. Specifically, for the results reported in Table 2 of our main paper, the inferior results for class prototype set 4 were due to poor local minimum while the inferior results for sets 1, 2, and 5 were due to instabilities.

Our experiments imply that too much class balancing can cause the training instabilities. We hypothesize that the model struggles to classify the unlabeled examples with lower quality class

Set	balance	Description	WD	LR	Δ	λ_u	τ	Accuracy (%)
1	3	Instabilities	8×10^{-4}	0.06	0	1	0.9	84 ± 6
1	3	Decrease λ_u, WD, LR	6×10^{-4}	0.04	0	0.5	0.9	89 ± 1
2	4	Instabilities	8×10^{-4}	0.06	0.25	1	0.95	80 ± 14
2	4	Decrease Δ, WD, LR	6×10^{-4}	0.04	0.1	1	0.95	94.5 ± 0.1
4	1	Local min	8×10^{-4}	0.06	0.25	1	0.9	77.5 ± 0.1
4	1	Increase Δ, τ	8×10^{-4}	0.06	0.3	1	0.95	93.2 ± 0.2
4	2	Local min	8×10^{-4}	0.06	0	1	0.9	81 ± 6
4	2	Increase λ_u	8×10^{-4}	0.06	0	2	0.9	92 ± 2
4	3	Local min	8×10^{-4}	0.06	0	1	0.9	81 ± 8
4	3	Increase λ_u	8×10^{-4}	0.06	0	2	0.9	88 ± 3
5	1	Instabilities	8×10^{-4}	0.06	0.25	1	0.95	86 ± 7
5	1	Decrease Δ	8×10^{-4}	0.06	0.1	1	0.95	90.7 ± 0.1
5	2	Instabilities	8×10^{-4}	0.06	0	1	0.9	89 ± 6
5	2	Decrease λ_u	8×10^{-4}	0.06	0	0.75	0.9	91.7 ± 1
5	3	Instabilities	8×10^{-4}	0.06	0	1	0.9	83 ± 10
5	3	Decrease WD, LR	6×10^{-4}	0.04	0	1	0.9	93.5 ± 2

Table 6: Illustration of the sensitivity to the hyper-parameters WD, LR, Δ, λ_u and τ. See the text for guidance on how to tune these hyper-parameters for situations with inferior performance due to instabilities or local minimums.

prototypes but the class balancing methods force the pseudo-labeling to mislabel samples in order to have the appearance of class balance. In these cases, it is better to reduce the amount of class balancing by using a smaller value for Δ for class balance methods 1 and 4, and using a smaller value for λ_u for class balance methods 2 and 3. In addition, we observed that decreasing weight decay (WD) and the learning rate (LR) improves performance when there were instabilities.

On the other hand, if the inferior performance is due to poor local minimum, increasing the amount of class balancing improves the performance. In this case, the accuracy increases and the standard deviation decreases by using a larger value for Δ for class balance methods 1 and 4, and using a larger value for λ_u for class balance methods 2 and 3. In addition, we observed that increasing weight decay (WD) and the learning rate (LR) improves performance. We also observed that it helps to increase τ if there are instabilities and to decrease τ in the poor local minimum situation. Table 6 provide examples of these recommendations.

Table 6 demonstrates how to improve the results presented in our main paper, where for consistency we used the same hyper-parameter values for all of the BOSS runs. Now we show that tuning can improve the test accuracies above the values reported in the paper.

Table 6 contains results of hyper-parameter fine tuning where we reported test accuracies below 85%. We list the class prototype set (set), the BOSS class balancing method (balance), weight decay (WD), initial learning rate (LR), the change in the confidence threshold for minority classes (Δ), the unlabeled loss multiplicative factor (λ_u), the confidence threshold (τ), and the final test accuracy in percent. Furthermore, we provide a short description that indicates if the training curve displays instabilities (i.e., the red curve in Figure 1) or a poor local minimum (i.e., the blue curve). Or the description points out the hyper-parameters that were tuned to improve the performance.

For example, the first row in the Table shows the results for set 1 using class balance method 3. Examination of the output from this run showed a curve resembling the red curve in Figure 1, implying the problem is one of instabilities. This calls for a decrease in λ_u, which improved the accuracy and reduced the standard deviation.

The other examples in Table 1 show improved results for both the problem of instability and for poor local minimums. The examples include modifying Δ, weight decay, learning rate, and τ. In most cases the final accuracies improved substantially with small changes in the hyper-parameter values, which demonstrates the sensitivity of one-shot semi-supervised learning to hyper-parameter values.