Meta Pseudo Labels

Hieu Pham [1 2] **Qizhe Xie** [1 2] **Zihang Dai** [1 2] **Quoc V. Le** [1]

Abstract

Many training algorithms of a deep neural network can be interpreted as minimizing the cross entropy loss between the prediction made by the network and a target distribution. In supervised learning, this target distribution is typically the ground-truth one-hot vector. In semi-supervised learning, this target distribution is typically generated by a pre-trained teacher model to train the main network. In this work, instead of using such predefined target distributions, we show that learning to adjust the target distribution based on the learning state of the main network can lead to better performances. In particular, we propose an efficient meta-learning algorithm, which encourages the teacher to adjust the target distributions of training examples in the manner that improves the learning of the main network. The teacher is updated by policy gradients computed by evaluating the main network on a held-out validation set.

Our experiments demonstrate substantial improvements over strong baselines and establish state-of-the-art performance on CIFAR-10, SVHN, and ImageNet. For instance, with ResNets on small datasets, we achieve 96.1% on CIFAR-10 with 4,000 labeled examples and 73.9% top-1 on ImageNet with 10% examples. Meanwhile, with EfficientNet on full datasets plus extra unlabeled data, we attain 98.6% accuracy on CIFAR-10 and 86.9% top-1 accuracy on ImageNet.

1. Introduction

Modern neural networks are often trained to minimize a cross-entropy loss. We can interpret this cross-entropy loss as the KL divergence from a *target distribution* over all the possible classes to the distribution predicted by a network. This interpretation arises a natural question: what should be this target distribution?

[1]Google Research, Brain Team [2]Carnegie Mellon University. Correspondence to: Hieu Pham <hyhieu@cmu.edu>.

Preprint.

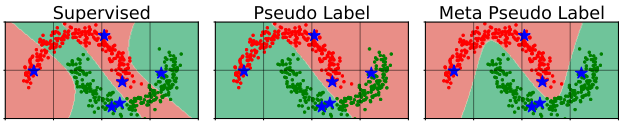

Figure 1: Conceptual behaviors of 3 methods on the TwoMoons dataset. There are 1000 red points and 1000 green points distributed onto two semicircles, out of which only 3 red points and 3 green points are labeled (the stars). A model can rely on both labeled and unlabeled points to find a classifier that best fits the data. The found classifiers are shown by the red and green regions. **Left:** Supervised learning with these 6 points leads to a wrong classifier. **Middle:** Pseudo label performs even worse than supervised learning because it relies on supervised learning to label the unlabeled data and in this case, supervised learning makes some mistakes in the top-left corner and the bottom-right corner of the figure. **Right:** Our method, Meta Pseudo Label (MPL), utilizes meta learning to train the pseudo labels throughout the course of the model's learning such that the student model will perform well on the 6 labeled examples. MPL finds a better classifier.

We argue that many, if not all, existing training algorithms for neural networks construct the aforementioned based on several heuristics. Specifically, in supervised learning, where neural networks are trained with labeled data, the target distribution is often a one-hot vector, or a smoothed version of the one-hot vector, *ie.,* label smoothing (Szegedy et al., 2016; Müller et al., 2019). In semi-supervised learning, the target distributions, also known as pseudo labels, are often generated on unlabeled data by a sharpened or dampened teacher model trained on labeled data, *eg.* Xie et al. (2019a); Berthelot et al. (2019). All such constructions for target distributions are heuristics that are designed *prior* to training, and thus they share an inherent weakness: they cannot adapt to the learning state of the neural networks being trained.

We propose to meta-learn the target distributions. In particular, we design a *teacher* model that assigns distributions to input examples to train the main model, which we henceforth refer to as the *student* model. Throughout the course of the student's training, the teacher observes the student's performance on a held-out validation set, and learns to generate target distributions so that if the student learns from such distributions, the student will achieve good validation performance. Since the meta-learned target distributions play the similar role to pseudo labels (Lee, 2013; Yarowsky, 1995; Riloff, 1996), we name our method *Meta Pseudo Label*

(MPL). MPL has an apparent advantage: the teacher model can adapt to the student's learning state and can improve the student's learning accordingly. Figure 1 demonstrates the behavior of MPL on the TwoMoons dataset. By adapting the target distributions to the student's learning state, MPL learns a better classifier than supervised learning and pseudo label.

Our experiments demonstrate substantial improvements over strong baselines and establish state-of-the-art performance on CIFAR-10, SVHN, and ImageNet. For instance, with ResNets on small datasets, we achieve 96.1% on CIFAR-10 with 4,000 labeled examples and 73.9% top-1 on ImageNet with 10% labeled examples. Meanwhile, with EfficientNet on full datasets plus extra unlabeled data, we achieve 98.6% accuracy on CIFAR-10 and 86.9% top-1 accuracy on ImageNet.

2. Motivations

In this work, we focus on training a C-way classification model parameterized by Θ, such as a neural network. Despite the wide spectrum of algorithms for training classification models, many of them can be summarized into minimizing the cross entropy between a *target distribution* $q_*(\mathbf{Y}|\mathbf{X})$ and the model distribution $p_\Theta(\mathbf{Y}|\mathbf{X})$, i.e.

$$\min_\Theta \mathcal{L}_{\text{CE}}(\Theta) = -\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\left[\sum_{c=1}^{C} q_*(c|\mathbf{x}) \log p_\Theta(c|\mathbf{x})\right]$$

Under this formulation, different algorithms simply correspond to specific instantiations of the target distribution:

- In *fully supervised training*, the target distribution is defined as the one-hot vector (single point distribution) representing observed / annotated value of the ground-truth class, i.e., for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{sup}}$, $q_*(\mathbf{Y}|\mathbf{x}) \triangleq \text{one-hot}(\mathbf{y})$.
- In *knowledge distillation* (KD; Hinton et al. (2015)), to compress the "dark knowledge" of a well trained larger model to a smaller one, for each data point $\mathbf{x}$, the predicted distribution of the large model $q_{\text{large}}(x)$ is directly taken as the target distribution, i.e. $q_*(\mathbf{Y}|\mathbf{x}) \triangleq q_{\text{large}}(\mathbf{Y}|\mathbf{x})$.
- In *semi-supervised learning* (SSL), a typical solution first employs an existing model q_ξ (trained on limited labeled data) to predict the class for each data point from an *unlabeled set*, and utilizes the prediction to construct the target distribution. There are two common versions:

$$\text{Hard label}: \quad q_*(\mathbf{Y}|\mathbf{x}) \triangleq \text{one-hot}\big(\operatorname*{argmax}_{\mathbf{y}} q_\xi(\mathbf{y}|\mathbf{x})\big)$$

$$\text{Soft label}: \quad q_*(\mathbf{Y}|\mathbf{x}) \triangleq q_\xi(\mathbf{Y}|\mathbf{x})$$

While these classic target distributions generally work well, recent works find they are often *not* the optimal choices. Instead, some heuristic methods have been exploited to slightly adjust the target distribution and lead to improved performance. Here, we review two notable examples.

Label smoothing It has been found that using the one-hot vector as the target distribution above in fully supervised machine translation and large-scale image classification such as ImageNet can lead to *overfitting*. To combat this phenomenon, label smoothing is proposed to smooth the one-hot distribution by allocating a small amount of uniform weights to all classes, i.e., for $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}_{\text{sup}}$, the target distribution is redefined as

$$q_*(c|\mathbf{x}) \triangleq q_{\text{smooth}}(c|\mathbf{x}) = \begin{cases} 1 - d + 1/C & \text{if } c = \mathbf{y} \\ d/C & \text{if } c \neq \mathbf{y} \end{cases}.$$

However, while label smoothing often helps at convergence, it also results in slower training.

Temperature Tuning For both KD and soft-label SSL, it has been found that explicitly introducing a *temperature* hyper-parameter to modulate the target distribution could be very helpful. Specifically, let $l_c(\mathbf{x})$ be c-th logit predicted by the teacher model, *eg.* the large model in KD and the existing model in SSL, then the target distribution is defined as

$$q_*(c|\mathbf{x}) \triangleq \frac{\exp\left(l_c(\mathbf{x})/\tau\right)}{\sum_{i=1}^{C} \exp\left(l_i(\mathbf{x})/\tau\right)},$$

where τ is the temperature that can be used to smooth ($\tau > 1$) or sharpen ($\tau < 1$) the distribution[1]. Intuitively, a smoother distribution could help to prevent overfitting or early mistakes in SSL. On the other hand, a sharper target could potentially speed up the training given it is correct.

From the success of these heuristic tricks, it is clear that how to construct the target distribution plays an important role in the algorithm design, and a proper method could lead to a sizable gain. Motivated from this observation, in this work, we focus on the construction of target distributions. In particular, instead of designing target distributions from scratch, we ask the question: whether there exists a *generic* and *systematic* method that can be used to modify the target distribution in an *existing algorithm* and lead to an improved target distribution and thus, to better performance.

As the first step towards this goal, we identify two intrinsic limits of many existing constructions:

1. The target distribution q_* is either chosen prior to training and then kept fixed afterwards or annealed/updated during training with an ad-hoc procedure;

[1] Note that as $\tau \to 0$, we can also recover the hard-label case.

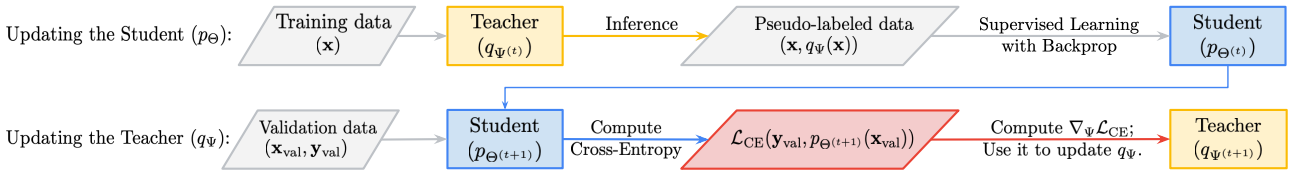

Figure 2: At each training step t, our Meta Pseudo Labels (MPL) update consists of two phases. **Updating the Student (top):** The teacher network q_Ψ assigns the conditional class distribution for a training example $\mathbf{x}$. The student p_Θ learns from $(\mathbf{x}, q_\Psi(\mathbf{x}))$ by standard supervised learning, updating from $\Theta^{(t)}$ to $\Theta^{(t+1)}$. **Updating the Teacher (bottom):** The teacher updates its parameters Ψ based on the resulting student's cross-entropy loss on validation data $(\mathbf{x}_{\text{val}}, \mathbf{y}_{\text{val}})$.

2. The modulation (smoothing or sharpening) of q_* does not depend on the data point in consideration.

Ideally, q_* should adapt to the *learning state* of p_Θ. For example, when the model is already confident enough for a data point at a time step, the target distribution may need to be smoothed to avoid overfitting this specific training instance. Figure 3 illustrates such an overfitting scenario from the perspective of train-validation discrepancy where the gradient computed using target distribution could push the student into a bad local minimum, which could be prevented by an alternate and noisier direction. With such motivation and intuition in mind, we next turn to our proposed method.

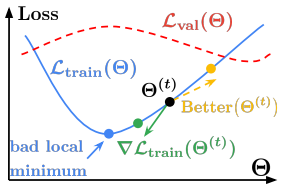

Figure 3: An illustration of manual correction of gradients. At $\Theta^{(t)}$ (black point), the model is near a bad local minimum (blue point). Both $\Theta^{(t)}$ and the local minimum have a low train loss but a high validation loss. If a human monitors this training process, they would suspect that $p_{\Theta^{(t)}}$ is overfitting and perhaps, would move Θ in a noisy direction (yellow dashed line). Meanwhile, the gradient vector (solid green line) pushes $\Theta^{(t)}$ even closer to the bad local minimum.

3. Meta Pseudo Labels

Our solution to the shortcoming of manually constructing the target distribution $q_*(\mathbf{x})$ is to *learn* $q_*(\mathbf{x})$ *throughout the course of training* p_Θ. In particular, we parameterize the target distribution $q_*(\mathbf{x})$ as $q_\Psi(\mathbf{x})$ and train Ψ using gradient descent. In Section 3.2, we describe two different parameterizations of q_Ψ. For now, it is sufficient to treat q_Ψ as a classification model, which assigns the conditional probabilities to different classes of each input example $\mathbf{x}$. We train q_Ψ based on the following principle:

> If Θ follows the gradients $\nabla_\Theta \mathcal{L}_{\text{CE}}(q_\Psi(\mathbf{x}), p_\Theta(\mathbf{x}))$ on training data $\mathbf{x}$, the resulting Θ *should* achieve a small validation loss $\mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_\Theta(\mathbf{x}_{\text{val}}))$.

Clearly, $q_\Psi(\mathbf{x})$ serves the same role as q_{large}, q_ξ, and $q_{\tilde{\Theta}}$ (Section 2), as $q_\Psi(\mathbf{x})$ provides the pseudo labels for p_Θ to learn. Due to this similarity, we follow the existing literature to call q_Ψ the *teacher model*, and call p_Θ the *student model*. Furthermore, the stated principle to optimize q_Ψ is essentially a meta-learning problem (see Appendix A), we name our method *Meta Pseudo Labels* (MPL).

3.1. MPL's Update Rules for Teacher and Student

As illustrated in Figure 2, each training step of MPL consists of two phases:

Phase 1: The Student Learns from the Teacher. In this phase, given a single input example $\mathbf{x}$, the teacher q_Ψ produces the conditional class distribution $q_\Psi(\mathbf{x})$ to train the student. We note that the input $\mathbf{x}$ does *not* need to come with any human-annotated label, as the teacher already computes its class-distribution $q_\Psi(\mathbf{x})$. The pair $(\mathbf{x}, q_\Psi(\mathbf{x}))$ is then shown to the student to update its parameters by back-propagating from the cross-entropy loss. For instance, if Θ is trained with SGD with a learning rate of η, then we have:

$$\Theta^{(t+1)} \triangleq \Theta^{(t)} - \eta \, \nabla_\Theta \mathcal{L}_{\text{CE}}(q_\Psi(\mathbf{x}), p_\Theta(\mathbf{x}))\big|_{\Theta^{(t)}} \tag{1}$$

Phase 2: The Teacher Learns from the Student's Validation Loss. After the student updates its parameters as in Equation 1, its new parameter $\Theta^{(t+1)}$ is evaluated on an example $(\mathbf{x}_{\text{val}}, \mathbf{y}_{\text{val}})$ from the held-out validation dataset, using the cross-entropy loss $\mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_{\Theta^{(t+1)}}(\mathbf{x}_{\text{val}}))$. Since $\Theta^{(t+1)}$ depends on Ψ via Equation 1, this validation cross-entropy loss is a function of Ψ. Specifically, dropping $(\mathbf{x}_{\text{val}}, \mathbf{y}_{\text{val}})$ from the equations for readability, we can write:

$$\begin{aligned} \mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_{\Theta^{(t+1)}}(\mathbf{x}_{\text{val}})) &\triangleq \mathcal{R}\Big(\Theta^{(t+1)}\Big) \\ &= \mathcal{R}\Big(\Theta^{(t)} - \eta \, \nabla_\Theta \mathcal{L}_{\text{CE}}(\boxed{q_\Psi(\mathbf{x})}, p_\Theta(\mathbf{x}))\big|_{\Theta^{(t)}}\Big) \end{aligned} \tag{2}$$

This dependency allows us to compute $\nabla_\Psi \mathcal{R}$ to update Ψ and minimize $\mathcal{R}\big(\Theta^{(t+1)}\big)$. This differentiation requires computing the gradient of gradient, which can be implemented by modern automatic differentiation frameworks such as TensorFlow (Abadi et al., 2016).

3.2. Instantiating the Teacher q_Ψ

While the student's performance allows the teacher to adjust and adapt to the student's learning state, this signal alone is *not* sufficient to train the teacher. In essence, the teacher observing the student's validation loss to improve itself is similar to an agent in reinforcement learning (RL) performing on-policy sampling and learning from its own rewards. Due to the potentially high sampling complexity, when the teacher has observed enough evidence to produce meaningful target distributions to teach the student, the student might have already entered a bad region of parameters.

A similar short-falling has been observed when training neural machine translation (NMT) models with RL (Bengio et al., 2015; Ranzato et al., 2016). Similar to MPL, RL training leads to better self-adaptive behaviors of NMT models. However, training with RL requires on-policy sampling from the NMT model, and hence would fail if the NMT model is not sufficiently trained a priori to produce reasonably correct samples. For this reason, NMT models must be trained in a supervised manner prior to being trained with RL (Bengio et al., 2015), or must be trained with a mixed signal from both RL and supervised learning throughout their courses of learning (Ranzato et al., 2016).

Here, we follow Ranzato et al. (2016) and add a supervised signal to MPL's teacher. In particular, at each training step, apart from the MPL updates in Equation 2, the teacher also computes a gradient $\nabla_\Psi \mathcal{L}_{\text{CE}}(\mathbf{y}, q_\Psi(\mathbf{x}))$ on a pair of *labeled* data $(\mathbf{x}, \mathbf{y})$. This gradient is then added to the MPL gradient $\nabla_\Psi \mathcal{R}$ from Equation 2 to update the teacher's parameters Ψ. In practice, we use the student's *validation* data to supervise the teacher, as illustrated in Figure 4. While the MPL algorithm interacts extensively with this so-called validation data, the student never directly learns from this validation set, effectively avoids overfitting. In fact, we observe no sign of overfitting in our experiments.

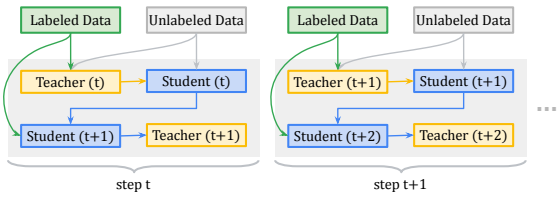

Figure 4: The MPL training procedure. At each step, the teacher receives both the MPL signal (Section 3.1) and the supervised signal from labeled data.

Adding the supervised signal to MPL introduces an implementation difficulty: we need to keep two classification models, the teacher and the student, in memory. While it is possible to train the pair of teacher-student with small architectures such as ResNets, for architectures with large memory footprints, *eg.* EfficientNet (Tan & Le, 2019), keeping two models limits the training batch size and leads to a slow training time. To allow training large models on large datasets, we design a more economical alternative to instantiate the teacher, termed ReducedMPL.

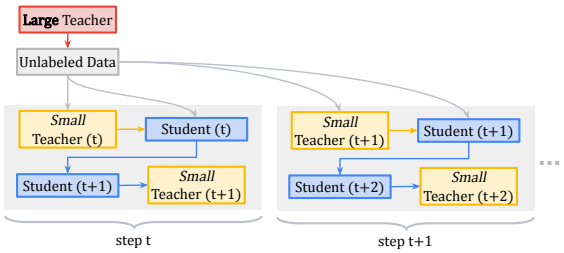

Figure 5: The ReducedMPL training procedure has 3 steps: (1) a large teacher q_{large} (red box) is pre-trained; (2) q_{large} assigns class distributions to the student's training data; (3) A small multi-layered perceptron q_Ψ calibrates the distributions computed by q_{large} to train the student. q_Ψ is trained along with the student, like the teacher in normal MPL.

In ReducedMPL, as shown in Figure 5, we first train a large teacher model q_{large} to convergence. Next, we use q_{large} to pre-compute all target distributions for the student's training data. Importantly, until this step, the student model has not been loaded into memory, effectively avoiding the large memory footprint of MPL. Then, we parameterize a reduced teacher q_Ψ as a small and efficient network, such as a multi-layered perceptron (MLP), to be trained the along with student. This reduced teacher q_Ψ takes as input the distribution predicted by the large teacher q_{large} and outputs a calibrated distribution for the student to learn. Intuitively, ReducedMPL works reasonably well because the large teacher q_{large} is reasonably accurate, and hence many actions of the reduced teacher q_Ψ would be close to an identity map, which can be handled by an MLP. Meanwhile, ReducedMPL retains the benefit of MPL, as the teacher q_Ψ can still adapt to the learning state of the student p_Θ.

4. Experiments

We demonstrate the effectiveness of MPL in two scenarios: 1) reduced datasets (Section 4.1): where limited labeled data is available, 2) full datasets (Section 4.2): where the full labeled data is used. In both scenarios, we experiment on CIFAR-10 (Krizhevsky, 2009), SVHN (Netzer et al., 2011) and ImageNet (Russakovsky et al., 2015). For experiments on full datasets, we use ReducedMPL due to the large memory footprint of MPL. Our goal is to experimentally confirm the benefit of MPL, which we re-emphasize as follows:

A teacher model is trained along with a student model to set the student's target distributions and adapt to the student's learning state.

4.1. Experiments with MPL on Reduced Datasets

We first compare MPL with existing semi-supervised learning algorithms on standard benchmarks with reduced datasets: CIFAR-10 with 4,000 labeled examples, SVHN with 1,000 labeled examples and ImageNet-10%.

Experiment Details. For CIFAR-10 and SVHN, we use a pre-activated WideResNet-28-2 (WRN-28-2) which has 1.5 million parameters (Zagoruyko & Komodakis, 2016). For ImageNet, we use a ResNet-50 which has 25.6 million parameters (He et al., 2016). We use 4,000 labeled examples from CIFAR-10, 1,000 labeled examples from SVHN, and roughly 128,000 labeled examples from ImageNet, which is approximately 10% of the whole ImageNet dataset. These images and their labels play two roles in our MPL training. First, they serve as *validation data* where the teacher measures the student's performance (Equation 2). Second, they are also the *labeled data* for the teacher (Figure 4).

Baselines. Our main baseline is Unsupervised Data Augmentation (UDA; Xie et al. (2019a)). We choose UDA as our main baseline for its state-of-the-art performance on the datasets and models in this section. UDA is a consistency regularization technique, which belongs to the category of semi-supervised learning (Section 2). In addition to UDA, we consider 3 other baselines: supervised learning, label smoothing, and RandAugment (Cubuk et al., 2019). Our goal here is to show that MPL can improve the performance of *all* these methods, hence further confirm the advantage of the adaptive teacher in MPL. We re-implement all baselines in our environment, and allocate the same amount of resources to tune hyper-parameters for all baselines. For each baseline, we compare the accuracy of the baseline with the accuracy of MPL's student, where the student learns from a teacher trained with the baseline algorithm plus the MPL signal. Further details are in Appendix C.

Results on CIFAR-10 and SVHN. In Table 1, we present our results with MPL on CIFAR-10 and SVHN, showing that MPL improves the accuracy of all baseline methods. For reference, we also include the results of a few other semi-supervised learning methods in the first block of Table 1. However, since these methods do not share the same controlled environment, the comparison to them is not direct, and should be contextualized (Oliver et al., 2018).

We observe that with 4,000 labeled examples for CIFAR-10 and 1,000 for SVHN, supervised training are prone to severe overfitting. Label smoothing and data augmentation, two of our baselines, are often utilized to reduce overfitting. From Table 1, we see that label smoothing improves the accuracy for SVHN, but fails to improve the accuracy of CIFAR-10. In contrast, MPL outperforms label smoothing on both datasets by about 1.5%. Meanwhlie, RandAugment (Cubuk et al., 2019) significantly improves the accuracy on both CIFAR-10 and SVHN, but MPL can further boost the accuracy by 2% on CIFAR-10 and by 0.4% on SVHN. Finally, MPL improves over UDA by 1.5% on CIFAR-10 and by 0.9% on SVHN. This improvement, along with the previous results, confirms our hypothesis about the benefit of MPL.

Methods	**CIFAR-10** (4,000)	**SVHN** (1,000)
Temporal Ensemble (2017)	83.63 ± 0.63	92.81 ± 0.27
Mean Teacher (2017)	84.13 ± 0.28	94.35 ± 0.47
VAT+EntMin (2018)	86.87 ± 0.39	94.65 ± 0.19
LGA+VAT (2019)	87.94 ± 0.19	93.42 ± 0.36
ICT (2019)	92.71 ± 0.02	96.11 ± 0.04
MixMatch (2019)	93.76 ± 0.06	96.73 ± 0.31
FixMatch (2020)	95.74 ± 0.05	97.72 ± 0.31
Supervised	82.14 ± 0.25	88.17 ± 0.47
Label Smoothing	82.21 ± 0.18	89.39 ± 0.25
Supervised+MPL	$\mathbf{83.71 \pm 0.21}$	$\mathbf{91.89 \pm 0.14}$
RandAugment (2019)	85.53 ± 0.25	93.61 ± 0.06
RandAugment+MPL	$\mathbf{87.55 \pm 0.14}$	$\mathbf{94.02 \pm 0.05}$
UDA (2019a)	94.53 ± 0.18	97.11 ± 0.17
UDA+MPL	$\mathbf{96.11 \pm 0.07}$	$\mathbf{98.01 \pm 0.07}$

Table 1: Image classification accuracy of WRN-28-2 on reduced CIFAR-10, SVHN. Higher is better. We report mean $\pm$ std over 10 runs. Results in the first block are taken from past papers for reference, while the rest shares the same environment and hyper-parameter settings.

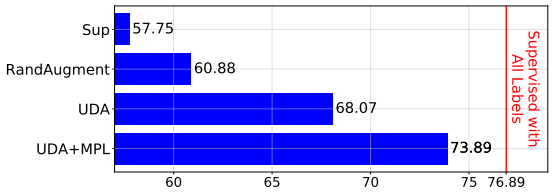

Figure 6: Top-1 accuracy of MPL and other methods on ImageNet-10%. MPL surpasses UDA by almost 6% while being only 3% below to training with all labels.

To our surprise, MPL even outperforms WRN-28-2 trained on all labeled examples from CIFAR-10 and SVHN. Specifically, on average, our WRN-28-2 achieves 94.9% accuracy on full CIFAR-10 and 97.4% on SVHN, which are lower than UDA+MPL's accuracy, as reported in the last row of Table 1. This means that UDA+MPL can be more than 10x efficient in terms of data complexity.

Results on ImageNet-10%. The gain of MPL here is even more significant than on CIFAR-10 and SVHN. As shown in Figure 6, MPL outperforms UDA by almost 6% in top-1 accuracy, going from 68.07% to 73.89%. To our knowledge, on ImageNet-10%, the only method that achieves a better accuracy than MPL is SimCLR (Chen et al., 2020a), which attains the top-1 accuracy of 74.4% on

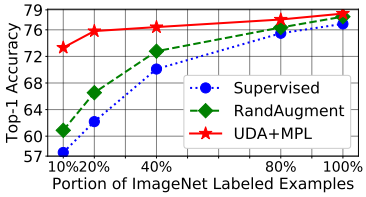

Figure 7: Top-1 accuracy of supervised learning, RandAugment, and UDA+MPL on ImageNet with 10%, 20%, 40%, and 80% of labeled examples.

ImageNet. However, SimCLR obtains this result using a 4x wider ResNet-50, while SimCLR's ImageNet-10% top-1 accuracy with the standard ResNet-50 is 65.6%, which is lower than our baseline UDA. Other strong SSL results on ImageNet-10% include: FixMatch's ResNet-50 top-1 accuracy of 71.5% (Sohn et al., 2020) and S4L's ResNet-50x4 top-1 accuracy of 73.2% (Zhai et al., 2019), all of which are lower than MPL's. There are also strong results from self-supervised learning such as MoCo (Chen et al., 2020b), but we do not compare against these results due to strong discrepancies in training procedures. These comparisons show that MPL sets the state-of-the-art performance on ImageNet-10% with ResNet-50. We also believe that MPL with larger architectures, such as ResNet-50x4, can further improve the method's performance.

MPL also continues to improve as more labeled data becomes available. In Figure 7, we further compare MPL to supervised learning and RandAugment on 20%, 40%, 80%, and 100% of the labeled examples in ImageNet. From the figure, it can be seen that MPL delivers substantial gains with less labeled data, but this gain dwindles as more labeled data becomes available.

4.2. Results with ReducedMPL on Full Datasets

To evaluate whether MPL can scale to problems with a large number of labeled examples, we now turn to full labeled sets of CIFAR-10, SVHN and ImageNet. We use out-of-domain unlabeled data for CIFAR-10 and ImageNet. We experiment with ReducedMPL whose memory footprint allows our large-scale experiments. We show that the benefit of MPL, *ie.,* having a teacher that adapts to the student's learning state throughout the student's learning, stil extends to large datasets with more advanced architectures and out-of-domain unlabeled data.

Model Architectures. For our student model, we use EfficinetNet-B0 for CIFAR-10 and SVHN, and use EfficientNet-B7 for ImageNet. Meanwhile, our teacher model is a small 5-layer perceptron, with ReLU activation, and with a hidden size of 128 units for CIFAR-10 and of 512 units for ImageNet.

Labeled Data. Per standard practices, we reserve 4,000 examples of CIFAR-10, 7,300 examples from SVHN, and 40 data shards of ImageNet for hyper-parameter tuning. This leaves about 45,000 labeled examples for CIFAR-10, 65,000 labeled examples for SVHN, and 1.23 million labeled examples for ImageNet. As in Section 4.1, these labeled data serve as both the validation data for the student and the pre-training data for the teacher.

Unlabeled Data. For CIFAR-10, our unlabeled data comes from the TinyImages dataset which has 80 million images (Torralba et al., 2008). For SVHN, we use the extra images that come with the standard training set of SVHN which has about 530,000 images. For ImageNet, our unlabeled data comes from the YFCC-100M dataset which has 100 million images (Thomee et al., 2015). To collect unlabeled data relevant to the tasks at hand, we use the pre-trained teacher to assign class distributions to images in TinyImages and YFCC-100M, and then keep K images with highest probabilities for each class. The values of K are 50,000 for CIFAR-10, 35,000 for SVHN, and 12,800 for ImageNet.

Baselines. We compare ReducedMPL to NoisyStudent (Xie et al., 2019b). NoisyStudent is a self-training approach (Section 2), which applies various regularization techniques to the student model. We choose NoisyStudent because it achieves a strong performance on ImageNet, and more importantly, because it can be directly compared to ReducedMPL. In fact, the *only* difference between NoisyStudent and ReducedMPL is that ReducedMPL has a teacher that adapts to the student's learning state.

Methods	CIFAR-10	SVHN	ImageNet
Supervised	97.18 ± 0.08	98.17 ± 0.03	84.49/97.18
NoisyStudent	98.22 ± 0.05	$\mathbf{98.71 \pm 0.11}$	85.81/97.53
ReducedMPL	$\mathbf{98.56 \pm 0.07}$	$\mathbf{98.78 \pm 0.07}$	**86.87/98.11**

Table 2: Image classification accuracy of EfficientNet-B0 on CIFAR-10 and SVHN, and EfficientNet-B7 on ImageNet. Higher is better. CIFAR-10 results are mean $\pm$ std over 5 runs, and ImageNet results are top-1/top-5 accuracy of a single run. All numbers are produced in our codebase and are controlled experiments.

Results. As presented in Table 2, ReducedMPL outperforms NoisyStudent on both CIFAR-10 and ImageNet, and is on-par with NoisyStudent on SVHN. In particular, on ImageNet, MPL with EfficientNet-B7 achieves a top-1 accuracy of 86.87%, which is 1.06% better than the strong baseline NoisyStudent. On CIFAR-10, MPL leads to an improvement of 0.34% in accuracy on NoisyStudent, marking a 19% error reduction.

For SVHN, we suspect there are two reasons of why the gain of ReducedMPL is not significant. First, NoisyStudent al-

ready achieves a very high accuracy. Second, the unlabeled images are high-quality, which we know by manual inspection. Meanwhile, for many ImageNet categories, there are not sufficient images from YFCC100M, so we end up with low-quality or out-of-domain images. On such noisy data, ReducedMPL's adaptive adjustment becomes more crucial for the student's performance, leading to more significant gain.

5. Analysis

Roadmap. We seek to understand the reasons for MPL's strong performance. First, in Section 5.1, we use mathematical reasoning to get an intuition of what MPL's teacher tries to achieve. However, as we shall explain, it is challenging to empirically observe our intuition on large-scale experiments. Instead, we provide empirical verification on a synthetic dataset, where our guess can be observed. Next, in Sections 5.2 and 5.3, we show some empirical behaviors of MPL on real datasets to *reject* two alternate and more trivial explanations of MPL's strong performance.

5.1. Hypothesis: MPL Fits the Validation Gradient

We revisit Equation 2 from Section 3:

$$\mathcal{R}\big(\Theta^{(t+1)}\big) = \mathcal{R}\Big(\Theta^{(t)} - \eta\, \nabla_\Theta \mathcal{L}_{\text{CE}}(q_\Psi(\mathbf{x}), p_\Theta(\mathbf{x}))|_{\Theta^{(t)}}\Big)$$

Denote $g(\Psi) \triangleq \eta \nabla_\Theta \mathcal{L}_{\text{CE}}(q_\Psi(\mathbf{x}), p_\Theta(\mathbf{x}))$. Under regulatory conditions, $g : \mathbb{R}^{\dim \Psi} \to \mathbb{R}^{\dim \Theta}$ is a smooth map. This allows us to differentiate $\mathcal{R}(\Psi)$ with respect to Ψ using the chain rule:

$$\begin{aligned} \nabla_\Psi \mathcal{R}\big(\Theta^{(t+1)}\big) &= \nabla_\Psi \mathcal{R}\Big(\Theta^{(t)} - g(\Psi)\Big) \\ &= J_\Psi(g)^\top \cdot \nabla_\Theta \mathcal{R}|_{\Theta=\Theta^{(t+1)}}\,, \end{aligned} \tag{3}$$

where $J_\Psi(g) \in \mathbb{R}^{\dim\Theta \times \dim\Psi}$ is the Jacobian matrix of g. Intuitively, this Jacobian quantifies how much a certain change in the teacher's parameters Ψ affects the student's training gradient. Thus, the product in Equation 3 quantifies how much the direction the teacher's parameter Ψ should change to align the student's training gradient with the student's validation gradient gradient. In other words, in expectation, the teacher encourages the student's training gradient $\nabla_\Theta \mathcal{L}_{\text{CE}}(q_\Psi(\mathbf{x}), p_\Theta(\mathbf{x}))$ to be similar to the student's validation gradient $\nabla_\Theta \mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_\Theta(\mathbf{x}_{\text{val}}))$.

This is a desired behavior, as we know that neural networks are over-parameterized models which are prone to overfitting on the training set, and to combat such degenerating behavior, we use the validation set for model selection and hyper-parameters tuning. MPL's behavior provides an end-to-end way to achieve a strong validation performance. Certainly, this behavior introduces a risk of overfitting to the validation set. However, as we will see in Section 5.2, this is not the case. We suspect that since the student never *directly* learns from the validation data, overfitting is avoided.

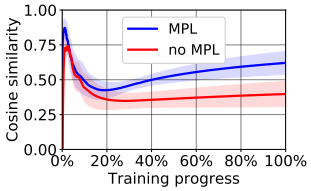

Figure 8: Cosine similarity of the student's gradient on *training* data and on *validation* data, measured throughout the course of training. It is clear that MPL improves this cosine similarity compared to supervised training.

In Figure 8, we plot the cosine similarity between these gradients on the synthetic dataset TwoMoons. Clearly, MPL gradually increases the similarity better than supervised learning. We *cannot* observe this phenomenon on experiments with large datasets, such as those in Section 4. This is because training on those large datasets requires stochastic gradient updates on minibatches of data, and stochastic gradients are poor estimates of the correct training gradient that MPL tries to make similar to the validation gradient. In fact, we observe that gradients on training and validation minibatches are almost uncorrelated, *ie.,* their cosine similarity are close to 0.

5.2. MPL is Not Label Corrections

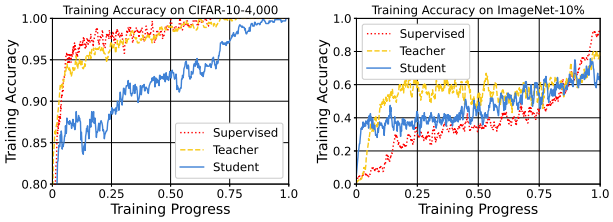

Figure 9: Training accuracy of MPL and of supervised learning on CIFAR-10 (4,000) and ImageNet (10%). Both the teacher and the student in MPL have lower training accuracy, effectively avoiding overfitting.

Since the teacher in MPL provides the target distribution for the student to learn and observes the student's performance to improve itself, it is intuitive to think that the teacher tries to guess the correct labels for the student. We empirically show that it is not the case. In Figure 9, we visualize the training accuracy of a purely supervised model, as well as of the teacher and the student model in MPLon CIFAR-10 (4,000) and ImageNet (10%). These accuracy are the result of taking argmax of the models' predictions on *validation* data throughout their training. As shown, the training accuracy of both the teacher and the student of MPL stay relatively low. Meanwhile, the training accuracy of the supervised model eventually reaches 100% much earlier. If MPL is simply performing label correction, then these accuracy should be high. Instead, we suspect that the teacher in

MPL is trying to regularize the student to prevent overfitting. This is the more appropriate behavior on small datasets like CIFAR-10 (4,000) and ImageNet (10%).

5.3. MPL is Not Only a Regularization Strategy

Figure 10: Target distributions that ReducedMPL computes throughout the course of training the student. For each image, the first column shows the distribution computed by a pre-trained model, while other columns show distribution computed by the teacher in ReducedMPL every quarter of the student's training process. Images are taken from the TinyImages dataset. The general pattern is that the distributions become more flat as the student is trained further, and we suspect this prevents overfitting in the student. However, there are exceptions, such as in the last row, where the distribution stays relatively sharp at the end.

In contrast to Section 5.2, one could think that MPL only injects noise to the student's learning to avoid overfitting. Here, we also negate this hypothesis. There are two ways for the teacher to inject noise to the student's learning: by flipping the target class, *eg.* tell the student the an image of a car is an image of a horse; or by dampening the target distribution. We empirically demonstrate that MPL's teacher follows neither pattern. In Figure 10, we visualize a few target distributions that a teacher model in ReducedMPL predicts for images from the TinyImages dataset. We observe two trends from the figure. First, the label with highest confidence for the images does not change at the quarters of the student's training process. This means that the teacher has not managed to flip the target labels. Second, the target distributions that the teacher predicts become *steeper* between 50% and 75%. As the student is learning during this time, if the teacher simply wants to regularize the student, then the teacher should dampen the distributions. Thus, we suspect that MPL is more than a regularization method.

6. Related Work

Synthetic Gradients. By letting the teacher generate the target distribution for the student model to learn, MPL equivalently lets the teacher determine the student's gradients. Learning the gradients belongs to a line of work called synthetic gradient (Andrychowicz et al., 2015). There are two major differences between MPL and synthetic gradient. First, MPL's gradient is restricted into a more specific subspace. In particular, the gradient $\nabla_\Theta \mathcal{L}_{\text{CE}}$ in MPL is computed from a cross-entropy, while synthetic gradients are computed based on intermediate representations of the student model, which has a much larger range of values. We suspect that such restriction makes the teacher of MPL provide more accurate gradients for the student model. Second, most work on synthetic gradient learn these gradients by regressing against the correct gradient, while MPL meta-learns the teacher to generate the student's gradients. An exception is "Learning Unsupervised Updates" (Metz et al., 2019), where the synthetic gradient is meta-learned via an explicit outer loop. Unlike MPL, Metz et al. (2019) has an explicit outer loop makes their training prohibitively expensive to scale to large datasets and large models like MPL.

Meta Learning. MPL shares the same goal with Meta Learning, *ie.,* to establish a positive bias that benefits the learning process of a sub-model (Bromley et al., 1994; Koch et al., 2015; Santoro et al., 2016; Finn et al., 2017; Liu et al., 2019). In MPL, this "bias" manifests via the target distribution of the training data for the student model. Similar to other meta-learning algorithms, MPL leverages the Jacobian-vector product (Townsend, 2017) to compute the "gradient of gradient" for MPL's teacher model (Equation 2, Section 3.1).

Semi-supervised Learning (SSL). Loosely speaking, SSL methods aim to utilize both labeled data and unlabeled data to train a model. As shown in our experiments (see Section 4), MPL makes use of both labeled and unlabeled data. Self-training and label propagation, which we discussed in details in Section 2, are SSL algorithms which assign class distributions to unlabeled data to extend the training dataset. In this sense, MPL is an SSL algorithm. However, a significant difference between MPL and other SSL methods is that our teacher model receives learning signals from the student's performance, and hence can adapt to the student's learning state throughout the course of the student's training. In Section 2 we have presented this motivation of MPL, and Section 4 we have empirically justified its benefit.

7. Conclusion

In this paper, we proposed Meta Pseudo Labels (MPL). Key to MPL is the idea that a teacher model can dynamically set the target distribution of training data for the stu-

dent to improve the student's learning. Experiments on CIFAR-10, SVHN, and ImageNet show that MPL significantly improves its corresponding baselines. Currently, MPL is too memory intensive for us to experiment on train large models and large datasets. However, we also proposed ReducedMPL which significantly reduces MPL's footprint, allowing us to verify the benefit of MPL's key idea in large scale experiment. As computational hardware rapidly develop, we believe that MPL will achieve better results.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., G, Derek . Murrayand Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. *Arxiv 1605.08695*, 2016. 3

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Frau, D., Schaul, T., Shillingford, B., and de Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, 2015. 8

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2015. 4

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., and Raffel, C. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*, 2019. 1, 5

Bromley, J., Guyon, I., LeCun, Y., Sickinger, E., and Shah, R. Signature verification using a "siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, 1994. 8

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, 2020a. 5

Chen, X., Fan, H., Girshick, R., and He, K. Improved baselines with momentum contrastive learning. *Arxiv, 2003.04297*, 2020b. 6

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical data augmentation with no separate search. *Arxiv 1909.13719*, 2019. 5

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017. 8

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *KDD*, 2017. 12

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 5

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *Arxiv, 1503.02531*, 2015. 2

Jackson, J. and Schulman, J. Semi-supervised learning by label gradient alignment. *Arxiv 1902.02336*, 2019. 5

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 12

Koch, G., Zemel, R., and Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In *International Conference on Machine Learning*, 2015. 8

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009. 4

Laine, S. and Aila, T. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*, 2017. 5

Lee, D.-H. Pseudo-Label: The simple and efficient semi-supervised learning method for deep neural networks. In *International Conference on Machine Learning Workshop*, 2013. 1

Liu, S. L., Davison, A. J., and Johns, E. Self-supervised generalisation with meta auxiliary learning. 2019. 8

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. Meta-learning update rules for unsupervised representation learning. In *International Conference on Learning Representations*, 2019. 8

Miyato, T., Maeda, S.-i., Ishii, S., and Koyama, M. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. 5

Müller, R., Kornblith, S., and Hinton, G. When does label smoothing help? In *Advances in Neural Information Processing Systems*, 2019. 1

Nesterov, Y. E. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 1983. 11, 12

Netzer, Y., Wang, T., Coates, Adamand Bissacco, A., Wu, B., and Ng, A. Y. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 4

Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., and Goodfellow, I. J. Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems*, 2018. 5

Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*, 2016. 4

Riloff, E. Automatically generating extraction patterns from untagged text. In *Proceedings of the national conference on artificial intelligence*, 1996. 1

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015. 4

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, 2016. 8

Sohn, K., Berthelot, D., Li, Chun-Liang Zhang, Z., Carlini, N., Cubuk, E. D., Kurakin, A., Zhang, H., and Raffel, C. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 5, 6

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. In *JMLR*, 2014. 12

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *CPVR*, 2016. 1, 12

Tan, M. and Le, Q. V. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, 2019. 4, 12

Tarvainen, A. and Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*, 2017. 5

Thomee, B., Shamma, D. A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., and Li, L.-J. YFCC100M: The new data in multimedia research. *Arxiv 1503.01817*, 2015. 6

Tieleman, T. and Hinton, G. RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 11

Torralba, A., Fergus, R., and Freeman, W. T. 80 million tiny images: a large dataset for non-parametric object and scene recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008. 6

Townsend, J. A new trick for calculating jacobian vector products, 2017. URL https://j-towns.github.io/2017/06/12/A-new-trick.html. 8

Verma, V., Lamb, A., Kannala, J., Bengio, Y., and Lopez-Paz, D. Interpolation consistency training for semi-supervised learning. In *International Joint Conference on Artificial Intelligence*, 2019. 5

Xie, Q., Dai, Z., Hovy, E., Luong, M.-T., and Le, Q. V. Unsupervised data augmentation for consistency training. *Arxiv, 1904.12848*, 2019a. 1, 5

Xie, Q., Luong, M.-T., Hovy, E., and Le, Q. V. Self-training with noisy student improves imagenet classification. *Arxiv 1911.04252*, 2019b. 6

Yarowsky, D. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pp. 189–196, 1995. 1

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *British Machine Vision Conference*, 2016. 5

Zhai, X., Oliver, A., Kolesnikov, A., and Beyer, L. S^4L: Self-supervised semi-supervised learning. *Arxiv, 1905.03670*, 2019. 6

Appendix for: Meta Pseudo Labels

A. Meta Learning Problem

We formally state the meta learning problem as mentioned in Section 3:

$$\text{[Outer loop]} \quad \Psi^* = \operatorname*{argmin}_{\Psi} \sum_{\mathbf{x}_{\text{val}}, \mathbf{y}_{\text{val}}} \mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_{\Theta_*}(\mathbf{x}_{\text{val}}))$$

$$\text{[Inner loop]} \quad \Theta_* = \operatorname*{argmin}_{\Theta} \sum_{\mathbf{x}} \mathcal{L}_{\text{CE}}(q_{\Psi}(\mathbf{x}), p_{\Theta}(\mathbf{x}))$$

We note that we do not directly solve this meta-learning problem, as the inner loop is prohibitively expensive to repeat for multiple times to train Ψ using gradient-based updates. Instead, MPL develops a step-wise strategy to update Ψ and Θ.

B. Generalized Update Rules of the Teacher

We demonstrate how to generalize the update rules of MPL to other training algorithms, such as Momentum (Nesterov, 1983) or RMSprop (Tieleman & Hinton, 2012). First, we revisit the teacher's MPL objective from Equation 2, which we rewrite below:

$$\mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_{\Theta^{(t+1)}}(\mathbf{x}_{\text{val}})) \triangleq \mathcal{R}\Big(\Theta^{(t+1)}\Big) = \mathcal{R}\Big(\Theta^{(t)} - \eta \, \nabla_{\Theta} \mathcal{L}_{\text{CE}}(\boxed{q_{\Psi}(\mathbf{x})}, p_{\Theta}(\mathbf{x}))\big|_{\Theta^{(t)}}\Big) \tag{4}$$

The dependency of the objective $\mathcal{R}\big(\Theta^{(t+1)}\big)$ on Ψ is through the student's gradient, namely the term boxed in magenta in the equation. Let us define:

$$f_{\Theta}(\Psi) \triangleq -\eta \nabla_{\Theta} \mathcal{L}_{\text{CE}}(q_{\Psi}(\mathbf{x}), p_{\Theta}(\mathbf{x})) \tag{5}$$

Then, Equation 4 can be rewritten as:

$$\mathcal{L}_{\text{CE}}(\mathbf{y}_{\text{val}}, p_{\Theta^{(t+1)}}(\mathbf{x}_{\text{val}})) \triangleq \mathcal{R}\Big(\Theta^{(t+1)}\Big) = \mathcal{R}\Big(\Theta^{(t)} + f_{\Theta^{(t)}}(\Psi)\Big) \tag{6}$$

This view allows us to generalize the computation of $\nabla_{\Psi}\mathcal{R}$ to arbitrary update rules by setting different forms for f. For example, for momentum update, we can simply set:

$$f_{\Theta}^{(\text{momentum})}(\Psi) \triangleq \mu m - \eta \nabla_{\Theta} \mathcal{L}_{\text{CE}}(q_{\Psi}(\mathbf{x}), p_{\Theta}(\mathbf{x})), \tag{7}$$

where μ is the momentum constant, typically set to 0.9, and m is the momentum vector, which does not depend on Ψ. Similarly, for RMSprop, we can set:

$$f_{\Theta}^{(\text{RMS})}(\Psi) \triangleq \mu m - \eta \cdot \frac{\nabla_{\Theta} \mathcal{L}_{\text{CE}}(q_{\Psi}(\mathbf{x}), p_{\Theta}(\mathbf{x}))}{\sqrt{(1-\rho) r + \rho \nabla_{\Theta} \mathcal{L}_{\text{CE}}(q_{\Psi}(\mathbf{x}), p_{\Theta}(\mathbf{x}))^2 + \epsilon}}, \tag{8}$$

where μ and ρ are the momentum and the RMS decay rate, m is the momentum and r is the moving average of squared gradients. Both r and m do not depend on Ψ.

In practice, to implement MPL, we create a shadow model of the student, whose variables are set to $f_{\Theta}(\Psi)$. We compute the gradient of the shadow variables of this shadow model, and then further back-propagate these gradients to Ψ.

C. Experiment Details

All our experiments are run on Tensor Processing Units, using slices of size 4x4, 8x8, or 16x16, depending on the experiment.

	Hyper-parameter	CIFAR-10	SVHN	ImageNet
Common	Weight decay	0.0005	0.0005	0.0002
	Label smoothing	0.0	0.0	0.1
	Batch normalization decay	0.99	0.99	0.99
	Number of training steps	1,000,000	1,000,000	500,000
	Number of warm up steps	2,000	2,000	1,000
Student	Learning rate	0.3	0.15	0.8
	Batch size	128	128	2048
	Dropout rate	0.35	0.45	0.1
Teacher	Learning rate	0.125	0.05	0.5
	Batch size	128	128	2048
	Dropout rate	0.5	0.65	0.1
UDA	UDA factor	1.0	2.5	16.0
	UDA temperature	0.8	1.25	0.75

Table 3: Hyper-parameters for MPL on reduced datasets in Section 4.1.

C.1. Details for Experiments in Section 4.1

Dataset Splits. For CIFAR-10 and SVHN, we download the datasets from their official websites, load them into `numpy_arrays`, and then select the first 4,000 and 1,000 examples, respectively. For ImageNet, we use the dataset shards preprocessed by Szegedy et al. (2016), which include 1,024 shards, and we take the first 102 shards, corresponding to 10% of all labeled data. This proceduure leads to a slightly imbalanced class distribution, *eg.* there are not exactly 400 images for each class of CIFAR-10 and not exactly 100 images for each class of SVHN. This is not our focus, and we use the same split for all controlled experiments – our baselines and our method MPL. The image resolutions are 32x32 for CIFAR-10 and SVHN, and are 224x224 for ImageNet.

Training Details. Both the teacher model and the student models are trained with Nesterov momentum (Nesterov, 1983), with a momentum constant of 0.9. We use the cosine learning rate schedule, starting a particular value and decaying to 0; the starting learning rate is a hyper-parameter. We also apply Dropout (Srivastava et al., 2014) at the prediction of *both* the teacher and the student. This means that when the teacher sets the target distribution for the student to learn, there are stochastic regularization.

Hyper-parameter Tunings. To select hyper-parameters, we reserve 400 labeled examples from the 4,000 labeled examples of CIFAR-10 and about 12,800 labeled examples from the 10% of labeled examples in ImageNet. For SVHN, since 1,000 examples are too few, we do not tune hyper-parameters; instead, we simply use the hyper-parameters found on CIFAR-10 for SVHN. We tune hyper-parameters using a contextual bandit optimizer, which is implemented by Golovin et al. (2017). We allow 256 trials for CIFAR-10/SVHN, and allow 128 trials for ImageNet. For hyper-parameter tuning, each trial is run for only 100,000 steps. Our tuning procedure is *incremental*. For example, we first tune hyper-parameters for training a supervised model, then when we tune for MPL, we use the found supervised hyper-parameters for the student in MPL and only tune the teacher's hyper-parameters. The optimal hyper-parameters are presented in Table 3.

C.2. Details for Experiments in Section 4.2

Training Details. Since our student models are EfficientNet, namely B0 for CIFAR-10 and SVHN and B7 for ImageNet, we simply use their corresponding hyper-parameters from Tan & Le (2019). Note that this means that our student models are updated with RMSprop, which necessitates the generalized update rules as described in Appendix B.

Our teacher model is a 5-layered multi-layered perceptron, with ReLU activation. This teacher model takes as input a probability distribution as predicted by our pre-trained model and returns a calibrated target distribution for the student to learn. We use the hidden size of 128 for CIFAR-10 and SVHN, and a hidden size of 512 for ImageNet. The teacher's parameters are updated with Adam (Kingma & Ba, 2015), using a learning rate of 0.0001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$. We do not need to tune this learning rate; we only try the log-range values, namely 0.1, 0.01, 0.001, and 0.0001, and use the largest learning rate that does not cause the teacher to get `NAN` values, which is 0.0001. We apply an L2 regularization of 10^{-4} to the teacher.