# Fine-grain Data Selection in Semi-supervised Learning

**Ching-Yi Lin, Xueyu Shi, Xianling Wang, Yijia Wang, Yangying Xu**

## 1 Introduction

Semi-supervised learning (SSL) deals with the classification problems with only a small proportion of the data labeled. The main idea is to make use of the adequate amount of unlabeled data during the training process [9, 2]. To successfully implement SSL, a primary assumption is the consistency in training set. Specifically, it assumes that adjacent inputs share similar labels (local consistency) [3], and that inputs with same structure have similar labels (global consistency) [1].

Among SSL methods, the self-training algorithm is a straightforward and commonly used one. According to our baseline experiments on the MNIST database, however, it provides inadequate improvement compared with supervised training. In this project, we seek to explore and improve the self-training process by modifying the data labeling selection process. On the other hand, deep nuerual network with self-ensembling model has shown promising results recently. We implement those approaches and compare them with self-training to show the effects of the usage for unlabeled data.

We demonstrate our models on the MNIST handwritten digits database. The MNIST dataset consists of 60,000 training images and 10,000 testing images. For each image of handwritten digit, the gray level of pixels is normalized and centered to $28 \times 28$ bounding box. This report is organized as follows. We describe our baseline model results in section 2, and review the classical self-training algorithms in section 3. Our modified self-training algorithms and other complimentary models are presented in section 4. The experiment results on the MNIST dataset are given in section 5. In section 6, we discuss the limitations as well as future steps of the self-training algorithm.

## 2 Background

For the baseline methods, we applied several supervised algorithms as well as the classical self-training on the MNIST dataset based on 6000 instances in total, with labeling rate varying from 0.1 to 0.5. Specifically, we investigated the performance of K-Nearest Neighbors (KNN), logistic regression, and deep neural networks (DNN) in both supervised classification and the self-training algorithm (Table 1).

In our results, compared with the supervised learning which only utilizes the labeled data, the self-training always gives better prediction accuracy. The DNN has the best performance in supervised learning, however, it gains the smallest improvement by the self-training process, which trains by incorporating unlabeled data. This may not be that surprising, since DNN is a more well-established model (in the aspect of amount of information it learns from the data) than KNN and logistic regression, and it already establish some "concepts" after fitting on the small amount of labeled data. With the adding-on of later-labeled data predicted by the model, these "concepts" can not change much.

Although the DNN benefits the least from self-training compared with the other two models, it still gives the best prediction accuracy over all models. It would be beneficial if we could improve the DNN-based self-training process. To overcome the failure to improve the established "concepts" in DNN, we consider to introduce model variability into the training process, and specifically, in the process how we select unlabeled data to be labeled. Another benefit of introducing model-wise variability is to alleviate the misleading from the wrong assumption of a certain model.

Table 1: The prediction accuracy on the test set with total 6,000 images (%)

| Label Rate | Method | DNN | KNN | Logistic |
|---|---|---|---|---|
| 0.1 | supervised | 86.07 | 81.20 | 83.85 |
| | self-training | 87.66 | 87.23 | 83.91 |
| 0.3 | supervised | 91.46 | 89.26 | 85.49 |
| | self-training | 91.85 | 92.37 | 85.77 |
| 0.5 | supervised | 93.44 | 91.68 | 86.80 |
| | self-training | 93.65 | 92.90 | 86.82 |

# 3 Related Works

## 3.1 Self-training

The self-training algorithm leverages both the labeled and unlabeled data, utilizing its own (confident) output of the unlabeled data instead of an observation [7]. Given a small labeled set and an unlabeled set, the classical self-training is implemented as following [9]:

1. Train a classifier from the labeled set only;
2. Calculate score/probability for each data in unlabeled set;
3. Select data in unlabeled set with score/probability higher than the threshold M;
4. Add those with higher confidence into training data, then go back to step 1.

The self-training is sensitive to wrong assumption in the assumed model. Also, a fixed threshold is used when defining the "confident output" in classical self-training. We make modifications to the self-training algorithm in the aspect of selecting data from unlabeled set to be imputed: i) As reasoned before, we applied different models for the data filtering process and the final classifier. We expect to lessen the potential harm of repeatedly using one model by introducing such variability. ii) Aside from introducing model variability, we pay attention to the definition of "confident prediction" in self-training algorithm, which depends the threshold above which we are confident to label the unlabeled data and add to the training process. If the threshold is too low, we will put back too many mis-classified data, which hurts the learning process. Modifying the classical self-training process by choosing an appropriate confidence threshold is another exploration we make in this report. Detailed illustrations of these modifications is presented in section 4.

## 3.2 DNN with Self-ensembling

Ensemble learning algorithm [6, 5] has shown success in improving the predictive performance of single neural networks-based classifiers in semi-supervised learning. Follows from the work from [5] (https://github.com/smlaine2/tempens), we investigate the effects of self-ensembling model for MNIST classification. We first explore the influence by using data augmentation ideas, dropout method and convolutional neural networks. Different network structures and parameter settings are then tested to study the strengths and limitations of DNN with self-ensembling model.

# 4 Method

## 4.1 Self-training

Self-learning essentially adds new label to the unlabeled data considering their predictive score/probability per epoch. This score/probability provides a good reference to how confident we are in this label assignment. This idea makes use of unlabeled data to strengthen the performance of the classifier. However, in our experiment, those added data seldom improved the accuracy a lot. There are two possible reason:

**1) Higher accuracy requirement from new-labeled data** Unlabeled data keeps being labeled and picked every epoch. Since this labeling is based on prediction, it is impossible that all the added labels are correct. With the score/probability threshold, new-labeled data usually has more confidence as well as higher classification accuracy than the test accuracy.

However, this relatively high accuracy is not enough to boost the accuracy of the classification task. Empirically, a 85%-accuracy model stays at the same accuracy after adding new data despite a 90% new-label accuracy, and lower new-label accuracy is worse. Those "good" accuracy is still very likely to contaminate the labeled set.

We provide the following explanation of this phenomenon. Conceptually, the ability of a labeled dataset considers both the size of data and how accurate the labels are. Labeled set can be regarded as a group of new labeled data with 100% new-label accuracy. With a strong classifier, this perfect labeled data may only result in 85-90% accuracy after training, showing that there exists a gap between label accuracy and classification accuracy. Due to this gap, the 90% new-label accuracy should not be expected to provide the same level of accuracy in the classification.

**2) Decreasing new-label accuracy along epochs** Another finding is the decreasing new-label accuracy from self-learning. Intuitively, the first epoch is able to select the most data, since there are numerous trivial, or easy-to-classified data in the unlabeled set originally.

Nevertheless, the more epochs we go, the classification is harder for the unlabeled data, since those easier ones have been picked in previous epochs. This increased difficulty not only shrinks the number of new labeled data, but also the quality. In our experiment, the first epoch usually picks same amount of data as the sum of all the rest. This uneven number of picked data creates a situation: After the first epoch, we are hardly able to find confident data from the rest in the unlabeled set.

From the observation above, we know that the original self-learning is prone to make a reckless pick with a great amount but low accuracy in each epoch. Those unlabeled data should be carefully chosen and labeled. Following this idea, we make modifications in self-training.

#### 4.1.1 Percentile threshold

In original self-training, data with score exceeding fixed threshold are picked. In addition to this fixed threshold, we apply a percentile threshold for every pick in an iteration. The picked data should not only exceed the given threshold above, but also need to be ranked in the top M% among the combination of labeled and unlabeled set. The hyper-parameter M here controls the number of data we pick for each epoch. Higher value picks less points in each turn, but provides a greater accuracy due to its high-ranking.

#### 4.1.2 Additional model for selecting unlabeled data

We adapt a different model than the final classifier to predict the unlabeled data here. The intuition comes from model ensemble technique, in which another model gives clues for those data hard to predict by the classifier model. In ensemble, it has been trusted that more diverse models could result in greater performance in classification. Thus, we want the model diversity has potential to easily label some unlabeled point which could be hard predicted by the classifier.

We proposed an algorithm based on the Gaussian Mixture Model (GMM) to predict the unlabeled set. This algorithm steps as follows.

1. Cluster from GMM: GMM splits the labeled set and the unlabeled set into k joint clusters. Each cluster has a single jointed mean and variance for both labeled and unlabeled set.
2. Find dominated cluster: For each cluster, we only focus on dominated cluster, which has one label with more than $T_D$ in the labeled set in this cluster. This $T_D$ is called "distribution threshold".
3. Select confident unlabeled data: We select the confident data in every dominated cluster, which weighted probability in GMM is higher than a given "distribution probability threshold.
4. Label the filtered data: Label those confident data with the label dominated in this cluster, and add them into labeled set. Go back to step 1.

**Cluster from GMM** GMM finds k clusters following Gaussian distributions. Fo each data point $\mathbf{x}$, a $k$-vector of weighted probability corresponding for each cluster is estimated. A data point is assigned to cluster $i$ if the $i$-th weighted probability is the highest. Formally,

$$\text{cluster}(\mathbf{x}) = \text{argmax}_i[N(\mathbf{x}|\mu_i, \Sigma_i)]$$

This clustering function can be treated as an one-to-one mapping from a data point to a cluster index. In the following step, we will apply cluster-level analysis to everyone. A suitable cluster will be selected as the candidate of new-labeled data.

**Find dominated cluster** Each cluster $C_i$, $i = 1, \ldots, k$ consists of both labeled data $(\mathbf{x}_j, y_j)$, $j = 1, ..., n_i, y_j \in \{0, 1, \ldots, 9\}$ and unlabeled $(\mathbf{x}_j, )$, $j = 1, ..., m_i$. The distribution of label will be used to evaluate the dominance. This distribution is defined by $P_{im} = \frac{1}{n_i}|\{j \mid y_j = m, j = 1, 2, \ldots, n_i)\}|$. This number represents the ratio of label $m$ in labeled data from cluster $i$.

The dominance is defined by the distribution of the most labal appeared in cluster $i$.

$$Dt_i = \max_m P_{im} = \frac{1}{n_i} \max |\{j \mid y_j = m, j = 1, 2, \ldots, n_i)\}| \leq 1$$

Here, the distribution threshold $T_D$, as a hyper-parameter, conveys the requirement for label purity. Clusters with mixed label couldn't be easily classified. Additional tricks like splitting into more cluster or going through another classifier might solve the problem and give more than one label into in-cluster data, but this is beyond the scope of this project. In here, we only pay attention to "pure cluster", namely the cluster i such that $Dt_i > T_D$. Higher threshold implies the dominated cluster is more pure in a specific label. In our intuition, this purity positively affects the new-label accuracy, but decreases the cluster selected every epoch.

**Select confident unlabeled data** Even the dominance of a cluster implies the high tendency to a specific label. Giving the label to every in-cluster data is a reckless behavior, as we have obtained in the classic self-training. To select those data closer or more belongs to the dominated cluster, we leveraged another filter by selecting only the data with higher weighted probability in GMM than a given threshold. This threshold is called "distribution probability threshold". It performs as a per-data-point control to the unlabeled set. Higher threshold shows that only those data close enough to the Gaussian mean are able to be given the label, and vice versa.

This distribution probability threshold acts the same role as the threshold in classic self-training. Described in section 3.1, the threshold M filters the higher scores obtained by the classifier. If we select the unlabeled pick model as the classifier's, this self-learning technique can be reduced into the baseline model we compare to.

**Label the filtered data** Applying both distribution threshold and percentile threshold, we can label those survived data confidently. In here, we have two available options. First, all of them are labeled regardless of the prediction from the classifier. Second, we can also label those data matching the prediction from the classifier. In the section 5, we will show the generalization performance of these two options.

### 4.2 Self-ensembling

In recent years, deep neural network (DNN) has been successfully used and become the state-of-art of supervised learning model in different areas, in particular when there have a large amount of training data. With the presence of small amount of labeled training data, regularization methods are always designed to avoid overfitting. Based on the work in [5], we implement two approaches that ensemble the neural network model to incorporate the unlabeled training data into the training process through dropout regularization method.

**Dropout regularization method** Dropout [8] is a randomized regularization method that the hidden units are ignored or "dropped out" randomly during evaluation and training, which deactivates all its incoming and outcoming connections in the network. By dropping the units out, the training process can be viewed as a large number of nueral networks with different architectures in parallel. The computation of the dropout method is extremely cheap and the resulted model is always robust and sparse through the past experience.

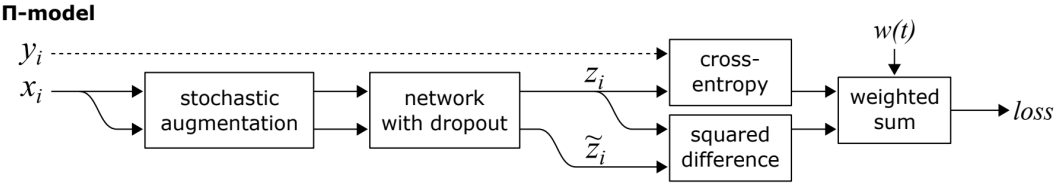



Figure 1: The structure of Π model.

**DNN with self-ensembling** We explore two approaches proposed in [5] that make use of dropout and unlabeled data. The reasons to use dropout method are twofold. One is to improve the over-fit of the model as we only have a small portion of training data that is labeled. On the other hand, in the supervised learning with dropout, although the existence of randomness in the final classifier, the two evaluations for the same input should be close with the "dark knowledge". In the proposed approach, we expect different predictions for the unlabeled data are close enough after the training process.

The structure of the first model, referred to as **Π model**, is shown in Figure 1. During the training process of Π model, to improve the variety of training data, data augmentation techniques are investigated in our experiments, including random crops and Gaussian noise. The loss function is formalized as follows:

$$\mathcal{L} = -\frac{1}{|L|}\sum_{i\in L} y_i \log(z_i) + w(t)\frac{1}{|D|}\sum_{i\in D}\left\|z_i - \tilde{z}_i\right\|^2,$$

where the training data $D$ consists of the labeled data $L$ and unlabeled data $U$. The first item of loss function $\mathcal{L}$ is the standard cross entropy that only consider the labeled data. The second item of loss function $\mathcal{L}$ is the mean square difference between two predictions $z_i$ and $\tilde{z}_i$ of all training data (labeled and unlabeled). We note that with the data augmentation method and dropout regularization, two evaluations of the same input would be different given the same network parameters. The time-related weight $w(t)$ is imposed with the second loss item to balance the effect between supervised loss and unsupervised loss.

Another idea to obtain prior evaluation $\tilde{z}_i$ is based on previous network predictions instead of a second evaluation. The structure of this idea is referred to as Temporal Ensemble (TE) model, whose model and loss function of TE are same with Π model. To estimate $\tilde{z}_i$, we use ensemble outputs $Z_i$ that

$$Z_i = \alpha Z_i + (1-\alpha)Z_i,$$

where $\alpha$ is the momentum parameter and compute $\tilde{z}_i = Z_i/(1-\alpha^t)$ with bias correction. Clearly, the training for TE model is faster than Π model as we only need one prediction for training data in every epoch.

**Setting $\mathbf{w(t)}$** At the beginning of training, $w(t)$ should be set to a small value, so that a "good" initial classifier can be obtained through supervised loss. The training algorithm is formalized in Algorithm 1.

**Data:** Training data $D$ = {labeled data $L$, unlabeled data $U$};
Weight function $w(t)$;
Neural network $f_\theta$ with random initial parameters $\theta$;
*[TE]* Updating parameter of the pseudo of unlabeled data, $\alpha \in [0, 1)$.
**Result:** Neural network parameters $\theta$.
*[TE]* Initialize $Z \leftarrow \mathbf{0}_{[N\times C]}$, $\tilde{z} \leftarrow \mathbf{0}_{[N\times C]}$.
**for** $t$ *in* $[1, max\_epoch]$ **do**
  **for** *each minibatch* $B$ **do**
    Make prediction $z_{i\in B}$ using NN $f_\theta$ with dropout.
    *[Π]* Make prediction $\tilde{z}_{i\in B}$ using NN $f_\theta$ with different dropout.
    $loss \leftarrow -\frac{1}{|B|}\sum_{i\in L\cap B} y_i \log(z_i) + w(t)\cdot\left\{\frac{1}{|B|}\sum_{i\in B}\left\|z_i - \tilde{z}_i\right\|^2\right\}$.
    Update $\theta$.
  **end**
  *[TE]* $Z \leftarrow \alpha Z + (1-\alpha)z$, $\tilde{z} \leftarrow Z/(1-\alpha^t)$.
**end**

**Algorithm 1:** Pseudocode for the Π-model and TE model. Lines starting with "*[Π]*" ("*[TE]*") only work in the Π-model (TE model, respectively), other lines work in both models.

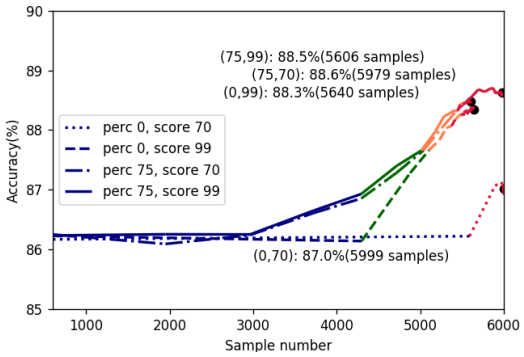



Figure 2: Performance of homogenoeous model in self-training

# 5 Result

## 5.1 Dataset

MNIST is a handwritten digit database and is widely used in evaluating classification problems. Our experiments contain two stages that use different sizes of MNIST. The first stage uses a small dataset for grid search of hyper-parameters. It consists of 600 labeled training data and 5,400 unlabeled data. The labeled data is identically-distributed among all classes. All the model measurements are tested on 10,000 testing set. The second stage uses a large dataset to verify our idea in a more practical dataset, the labeled training set and testing set are same as the former. The only difference is the size of the unlabeled dataset, increased from 5,400 to 59,400, which is the most data we can get from this database.

## 5.2 Homogeneous Model in Self-training

We first evaluate the effect of the percentile threshold. As the setting with percentile threshold 0 and score/probability M can be reduced into the classic self-learning, we only perform the grid search of these two parameters.

### 5.2.1 Percentile threshold

Figure 2 shows how test accuracy and number of label increases as the epoch goes. Here we can have some observations.

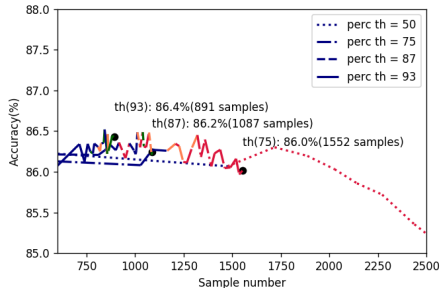
1. The baseline of self-training (percentile threshold = $0\%$) pick a large number of data in the first epoch. The first portion of the line shows this number clearly. Cases with score threshold 70 and 99 have 4300 and 5600 labeled data after the first epoch respectively. In contrast, applying percentile threshold ($75\%$) shrinks the number into 1350 after first epoch.
2. The quality of unlabeled pick becomes worse and worse. This phenomenon also happens in heterogeneous case. We will give more discussion in 5.3.

## 5.3 Heterogeneous Model in Self-training

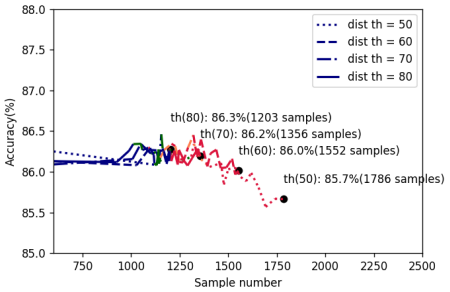
Heterogeneous model uses a novel structure in self-training task. We have two hyper-parameter can be configured in GMM-based unlabeled pick. In here, we fix one of them and sweap the other to see the change in performance.

### 5.3.1 Distribution threshold

As the intuition, the higher distribution threshold leads fewer added sample after 20 epochs. However, with the higher quality, the training with distribution threshold 87 and 93 suffers less from the bad new-label quality in the last epochs. But for the other cases, a series of orange and red lines in the tail shows the worse quality after several epochs. Furthermore, they affect the test accuracy more or less.




(a) Percentile threshold
(distribution probability threshold 60%)




(b) Distribution probability threshold
(percentile threshold 75%)

Figure 3: Performance of heterogeneous model in self-training

### 5.3.2 Percentile threshold

Fixing percentile threshold, the distribution threshold shows less effect in result than the previous. Most of them follow the same pattern in the beginning, but different in the last epochs. A lower probability threshold is still able to pick something but low quality unlabeled data. In contrast, the opposite case is slow to choose new labeled data from the rest. This matches our hypothesis: Easy data is keen to be picked in the first few epochs, and those harder data will be left.

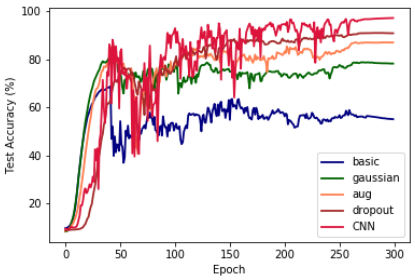
## 5.4 Ensembled DNN Exploration

**Experiment setup.** The networks are trained using Adam [4], where the maximum learning rate $\lambda_{max} = 0.001$, Adam momentum parameters are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and maximum epoches is 300. In $\Pi$ model, we use $w_{max} = 30$, and for TE model, $w_{max}$ is set to 50.

**Preliminary results for network structures.** Our preliminary experiments focus on the effects of data augmentation techniques, dropout and convolutional neural network (CNN). The DNN structure is 3 dense hidden layers with units 256, and 2 dropout layer with $p = 0.5$. The network structures and different settings are shown in Figure 4 (a), where "augment" means the random crops, "noise" means Gaussian noise with standard deviation 0.15. In Figure 4 (b), the test set accuracy suggest that the data augmentation techniques significantly improve the variety of training data. We can achieve 90.86% and 97.2% test accuracy for dropout and CNN, respectively. It is not a surprising result due to the fact that CNN is able to capture the space invariant and shift invariant.

| DNN | basic | GN | aug | dropout | CNN |
|---|---|---|---|---|---|
| augment | | | X | X | conv32+64+2by2MP |
| noise | | X | X | X | dropout0.5 |
| dense256 | X | X | X | X | conv32+64+2by2MP |
| dropout0.5 | | | | X | dropout0.5 |
| dense256 | X | X | X | X | conv 32 |
| dropout0.5 | | | | X | nonlinear256 |
| dense256 | X | X | X | X | Global-pool |

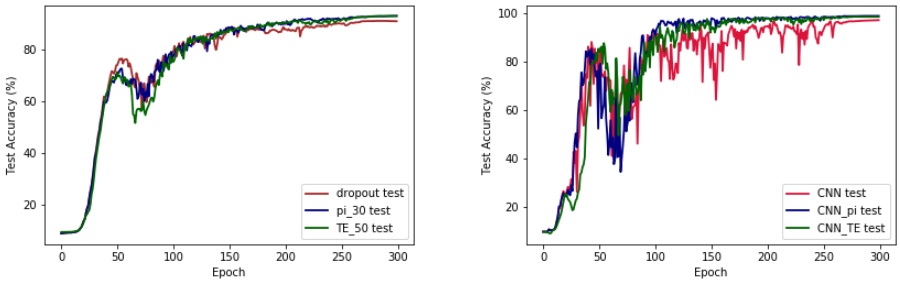
(a) Model structure




(b) Results

Figure 4: Prediction accuracy for stochastic augmentation, dropout and CNN.

In Figure 6, we test $\Pi$ model and TE model in DNN and CNN structure. With the used of 600 labels, we observe 2.04% and 2.02% accuracy improvements over DNN structure by applying $\Pi$ and TE model. Also for CNN structure, we can achieve 98.77% and 98.61% test accuracy for $\Pi$ and TE model, respectively.




(a) DNN (b) CNN

Figure 5: Prediction accuracy with different network structure.

### 5.5 Generalized to the Whole MNIST

After we observe the effect of introduced threshold, we expand the dataset from 6000 data to the whole MNIST. Each case in the following still has only 600 labeled data, but the number unlabeled data increases from 5,400 to 59,400.

Table 2: The prediction accuracy on the test set with total 6,000 images (%)

| Self-training | | | | Self-ensembling | | |
|---|---|---|---|---|---|---|
| DNN | Baseline[7] | Heterogeneous | Homogeneous | CNN | CNN+Π | CNN+TE |
| 86.07 | 88.12 | 88.94 | 86.26 | 97.20 | 99.18 | 99.07 |

## 6 Discussion

### 6.1 Limitation of Self-training

Self-training labels a set of unlabeled data in each epoch and includes them in next training iterations. We assume that the increased quality improve the generalization performance. However, the limited labeled data can only give limited knowledge for the model. In our experiments, we only obtain 2-3% accuracy improvement from the baseline model.

### 6.2 Unlabeled Data Usage Self-training and Self-ensembling

In this project, we exploited two techniques to provide more knowledge from unlabeled data. Essentially, self-training and self-ensembling have distinct idea in usage of those unlabeled data. In self-training, labeling or not is a hard assignment. The new labeled has the same importance as the original data, but the data hasn't been labeled cannot contribute anything during the training process. In contrast, self-ensembling treats all the unlabeled data in uniform, trying to minimize the mean square difference under dropout regardless of the correctness or classification difficulty of the unlabeled data.

### 6.3 Additional Information from Data Augmentation

Our experiment shows the best we can do for the limited number of data without any modification on data. To increase more information, we may leverage data augmentation in the future. Figure 5 shows the benefit of the data augmentation in self-ensembling. If we can put more information by exploiting these methods, we might further improve the generalization performance in the end.

## References


[1] Mikhail Belkin and Partha Niyogi. Semi-supervised learning on riemannian manifolds. *Machine learning*, 56(1-3):209–239, 2004.

[2] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010.

[3] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. Cluster kernels for semi-supervised learning. In *Advances in neural information processing systems*, pages 601–608, 2003.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv:1610.02242*, 2016.

[6] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in neural information processing systems*, pages 3546–3554, 2015.

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

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

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