Semi- Supervised Learning for Fine- Grained Classification with Adaptive Pseudolabeling

Ishita Chakravarthy
ichakravarth@umass.edu

Matthew James
matthewjames@umass.edu

Mihir Thalanki
mthalanki@umass.edu

Abstract

Fine- grained image classification tasks require models to pick up on subtle distinctions in images, with lesser labeled data. Semi-supervised learning with pseudo-labeling can work better by combining small labeled sets with large unlabeled ones. Pseudo-labeling approaches which use a fixed threshold have a variety of drawbacks, which motivates the use of adaptive thresholds. This paper aims to analyze the use of adaptive global and class-specific thresholds on the Fungi dataset, and compare the results of Top 1 and Top 5 accuracies.

1. Introduction

Fine-grained image classification tasks are particularly harder as compared to basic classification tasks due to lack of annotated data, along with the need for models to pick up on subtle distinctions in images. Semi-Supervised learning can be used for the above task, since it leverages a smaller number of classes along with a larger set of unlabeled data. Pseudo-labeling [11] is a common approach taken to label larger sets of unlabeled data, by selecting a threshold to identify reliable predictions, which are then utilized to enhance the learning process. The project aims to look at utilizing adaptive thresholds to overcome some of the challenges cause by fixed threshold pseudo-labeling in the task of fine-grained classification.

1.1. Motivation

For pseudo-labeling [11], a fixed threshold is used to create pseudo-labels, and then train a model. The threshold significantly impacts the model's learning process and quality of labels available to the model. Low thresholds would lead to images getting assigned wrong pseudo-labels, and higher thresholds would exclude examples from contributing to the model.

This motivates the approach of using a moving threshold across iterations.

Our approach is to have a global threshold that iteratively decays during the training process. This decay is crucial because, in the early stages, a higher threshold ensures that only the most confident predictions are used, which reduces the chances of incorrect pseudo-labels propagating through the training process. Over time, as the model becomes more reliable, the threshold is reduced to allow more predictions to contribute to training.

Additionally we propose using class-specific thresholds as it is suboptimal to rely on one threshold to handle the varying characteristics, representation and difficulty of different classes. A global threshold may disproportionally favor overrepresented classes by generating more pseudo-labels for these classes. Class-specific thresholds enable the model to handle each class separately and can adapt the learning to the model's confidence over different classes and it's representation in the dataset.

1.2. Dataset

The dataset used is Fungi [3] taxonomy dataset. The dataset is around 15GB in size, and the class distribution is long tailed. Long-tailed distributions is common in the domain of fine-grained classification, as the common classes will have more examples and the rarer classes will have lesser data available. Table 1 provides details on the number of images in each, number of labeled and unlabeled data, and class distribution for the two datasets.

Dataset Component	Semi-Fungi
Labeled Data	4141
Unlabeled Data	13166
Validation Data	4000
Test Data	4000
Number of Classes	200

Table 1. Summary of Dataset Components

2. Related work

2.1. Semi-Supervised Learning

Semi-supervised learning (SSL)[7] has seen a steady increase in attention over time due to its ability to leverage both labeled and unlabeled data into model training. There have been several different categories of SSL techniques which have grown in relevance in recent years. Some of these categories include self-training, consistency-based learning, and self-supervised learning.

Self-Training[12] is an iterative process, where we use the model's predictions to automatically generate labels for unlabeled data which have been classified with a high confidence score. A popular technique in this category is pseudo-labeling[11], which uses the predictions made on unlabeled data with a confidence score above a certain threshold, towards training the model. Our experiments in this paper will be focused on this approach to semi-supervised learning.

Another variant of pseudo-labeling is curriculum pseudo-labeling[6]. In this method, instead of using all pseudo-labeled data indiscriminately, the samples are organized by confidence levels. The idea is to start training with samples that have high-confidence pseudo-labels, which are more likely to be correct, and then gradually introduce samples with lower confidence. In both these methods, a fixed threshold of some form is used to determine our pseudo-labels in an effort to reduce confirmation bias.

Consistency-based learning methods learn by encouraging consistent predictions by the model on unlabeled data. Typically, this would be performed on varying augmentations of the data[2][10][14]. Some popular approaches in this category include MixMatch[4], FixMatch[15], FlexMatch[20], UDA[17].

The use of self-supervised learning tasks to improve semi-supervised learning is another popular approach[19]. This involves using surrogate (pre-text) tasks that can be formulated using only unsupervised data, such as predicting the order of patches and image rotations, during semi-supervised learning [8][13].

2.2. Thresholds in Pseudo-Labeling

To mitigate confirmation bias in pseudo-labeling[1], confidence-based thresholding techniques have been introduced to maintain pseudo-label quality by retaining only unlabeled data with confidence scores exceeding a set threshold. UDA[17] and FixMatch[15] apply a static, predefined threshold throughout training, while FlexMatch[20] varies the threshold on a class-by-class basis, adjusting according to each class's learning progress, as estimated by the count of confident unlabeled samples. Dash[18] sets a threshold based on labeled data loss, adjusting it via a fixed mechanism. AdaMatch[5], a more recent method, seeks to unify SSL and domain adaptation by using a predefined threshold scaled by the average confidence of labeled data batches to filter out noisy pseudo-labels. Although AdaMatch applies distribution alignment to promote fair predictions on unlabeled data, it also depends on a predefined threshold and may overlook the unlabeled data distribution when labeled samples are too limited to represent it adequately. These thresholding approaches may fail to choose optimal thresholds due to limited consideration of the relationship between model learning status and threshold settings, underscoring the need for a more self-adaptive thresholding strategy in pseudo-labeling.

3. Method

The proposed implementation is a novel pseudo-labeling variant that dynamically adjusts the threshold during training. The algorithm utilizes a global threshold and class-specific thresholds, to assign pseudo-labels. Initially, the model receives a set of labeled L_{in} and unlabeled data U. At each iteration, a global threshold and class thresholds are computed to reflect the overall and class-specific confidence levels. A combined threshold is calculated using the harmonic mean of the global and class-specific thresholds, ensuring a balance between generalization and class-level precision. An unlabeled sample is assigned a pseudo-label if the maximum softmax probability meets the confidence criteria defined by the combined thresholds. The loss for each iteration is computed using the labeled data and the generated pseudo-labels.

An initial approach to implement a dynamic global threshold involves using a linearly or exponentially decaying threshold over iterations. A more refined method updates the threshold iteratively by employing a formula based on previous iterations threshold and current confidence of the model. For class-specific thresholds, a naive implementation computes a threshold for each class based on the frequency of that class in L_{in} and the model's confidence in predicting that class.

RESNET-18 was used for our experimental setup. Section 4.5 talks about the comparision between using RESNET-18 and RESNET-50.

3.1. Global threshold

As mentioned above, the initial approach to setting a global threshold involved using a linearly decaying threshold. The global threshold G_i for iteration i was computed by performing a linear decay over the range [0.80, 0.95] over 10000 iteration based on the formula:

$$G_i = 0.95 - i(1.5 * 10^{-5})$$

This linear decay ensured stricter confidence criteria at the start of training and more leniency as training progressed. A more adaptive strategy for moving the threshold using the model's confidence used an iterative formula. [16] also utilizes moving global thresholds in a similar approach, however, they use increasing thresholds. Our experiments with a linearly increasing threshold yielded results that were not promising. At the early stages of training, when the model has not learned significant information yet, a low threshold led to incorrect pseudo-labeling. These errors propagated through the training, especially to underrepresented classes, hence degrading overall performance.

Based on these insights, experiments were run to adopt and refine a formula for decaying thresholds. G_i was initialized to 0.95 with a minimum value set to 0.8, to ensure that it does not drop excessively low. The decay rate $(1 - \lambda)$ was set to 0.001, to ensure incremental changes to the threshold at each iteration. The threshold was adjusted dynamically based on the model's overall confidence (softmax probability of the predicted class). Specifically, whenever the model's overall confidence increased, the threshold was updated proportionally to the change in confidence $\Delta_{confidence}$ using:

$$G_i = G_{i-1} - (1 - \lambda)(\Delta_{confidence})$$

3.2. Class threshold

The initial experiments used a naive approach for setting class thresholds as directly proportional to frequency of the class. For a class c, the threshold for iteration i was given by:

$$C_i(c) \propto f(c)$$

where $f(c)$ represents the frequency of class c in L_{in}. Once the class frequencies were calculated, the thresholds were normalized using the below equation to ensure that class thresholds lie between the range of [0.8,0.95]. Specifically, the class with the highest class frequency received a class threshold of 0.95, and lowest received a threshold of 0.8.

$$C_{norm}(c) = \frac{C_i(c) - min(C_i)}{max(C_i) - min(C_i)}$$

$$C_{i_{adjusted}}(c) = C_{norm}(c) * 0.15 + 0.8$$

For the iterative approach, class-specific thresholds were updated similar to the global threshold detailed above. The initial values were based on the frequency-based class thresholding approach outlined above, which demonstrated promising results. To prevent the thresholds from dropping excessively low, a minimum threshold of 0.8 was enforced. The class-specific thresholds were updated using:

$$C_i(c) = C_{i-1}(c) - (1 - \lambda)(\Delta_{confidence})$$

3.3. Combining global and class based thresholds-FIX

[16] finds a value of threshold per class using global and local thresholds to ensure that all information is used when generating a mask. The final values were obtained multiplying the two values of global threshold and class thresholds. However, based on our experiments, this produced poor results, because the thresholds dropped significantly. For example, when global threshold is 0.8 and a class-specific threshold is 0.8, the final threshold for the class decreases to 0.64. This results in a significant drop in the model's performance. Therefore, a simple multiplication is not ideal.

To address this, we used a harmonic mean to combine the global and class-specific thresholds. The harmonic mean is particularly suited for this task because it places more emphasis to the smaller threshold value, preventing it from dropping too low. The harmonic mean balances the two threshold while maintaining the given range of [0.8, 0.95].

$$T_i(c) = \frac{2 * C_i(c) * G_i}{C_i(c) + G_i}$$

4. Results

4.1. Motivation for results

4.1.1 Pseudo-labeling with different thresholds

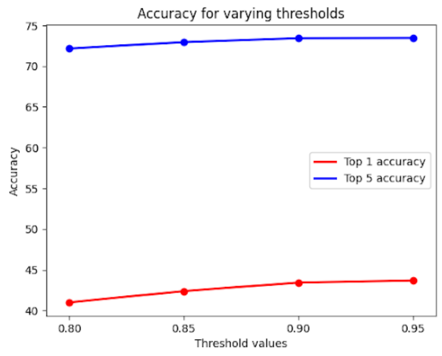

Figure 1. Model Accuracy across pseudo-labeling thresholds

Fig 1 shows the impact of varying pseudo-labeling thresholds $\{0.8, 0.85, 0.9, 0.95\}$ on the Semi-Fungi dataset for ResNet-18 model with L_{in} and U_{in} data. For the rest of the results and experiments, the baseline of a static threshold of 0.8 was considered. The results illustrate the sensitivity to the threshold choice and emphasizes the importance of threshold optimization. A higher threshold requires a higher confidence level for a pseudo-label to be considered, which influences the number and quality of the pseudo-labels generated.
This process is more challenging than standard hyper-parameter tuning because fine-grained class distinctions make it hard to find a single threshold that works across all classes. We investigate this further in the next section.

4.1.2 Insights from Class-Wise accuracies

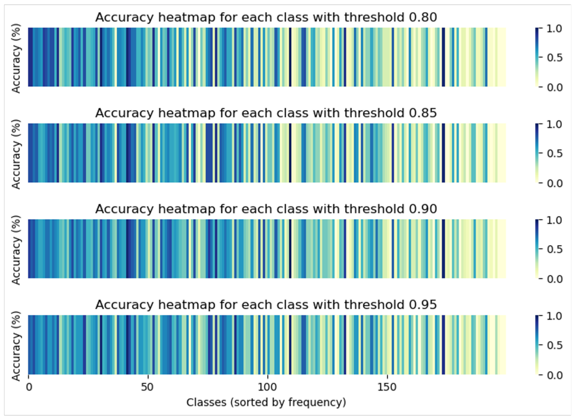

Figure 2. Model Accuracy across pseudo-labeling thresholds

Fig 2 shows the model's performance across the 200 classes (sorted by frequency) as an accuracy heatmap for 4 thresholds. The color intensity in the heatmap corresponds to the accuracy within that class. A common trend seen across thresholds: classes that are more frequent in the training dataset L_{in}, exhibit higher accuracy compared to those that are under-represented. Additionally, the choice of threshold not only impacts the overall model performance but also the accuracy achieved within individual classes. Some classes exhibit significant variance in accuracy across thresholds, indicating that using the same threshold for all classes might not be the best strategy.

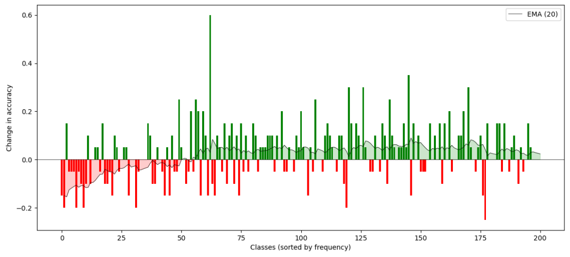

Figure 3. Change in accuracy from threshold 0.8 to 0.95 across classes. Green bars indicate better performance and red bars indicate worse performance for the corresponding class

Fig 3 illustrates the impact of adjusting the threshold from 0.8 to 0.95 on class-wise accuracy. Each bar represents the accuracy difference for the corresponding class from a threshold change of 0.8 to 0.95. The variation seen across these classes underscore the fact that not all classes respond similarly to threshold changes. Frequent classes tend to lose performance at higher thresholds due to stricter confidence requirements, which may exclude valuable data. Conversely, under-represented classes benefit, as the model prioritizes reliable labels from these classes. With a threshold that dynamically updates, the advantages of low and high thresholds can be leveraged in training the model. The following sections present the results from experiments utilizing this adaptive thresholding approach.

4.2. Adaptive global Threshold Results

4.2.1 Linearly decaying global threshold

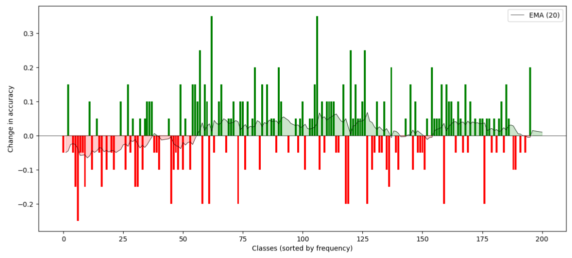

Figure 4. Change in accuracy across classes from static threshold of 0.8 to linearly decaying threshold from [0.8, 0.95]. Green bars indicate better performance and red bars indicate worse performance for the corresponding class

The experiment above used a linearly decaying threshold, as detailed in 2.2, with the threshold being decayed by a small amount every iteration in the range of [0.8,0.95]. Figure 4 highlights how decaying the threshold affects class accuracy based on their frequency. Underrepresented classes show an improvement in accuracy with the linearly decaying threshold. However, this comes with the trade-off in accuracy among the overrepresented classes.

For underrepresented classes, the model learns better with a higher threshold. Decaying this threshold ensures that the model has higher performance than 0.8 for these. For overrepresented classes, a high threshold means poorer performance. Decaying the threshold does not improve accuracy.

4.2.2 Iteratively decaying global threshold

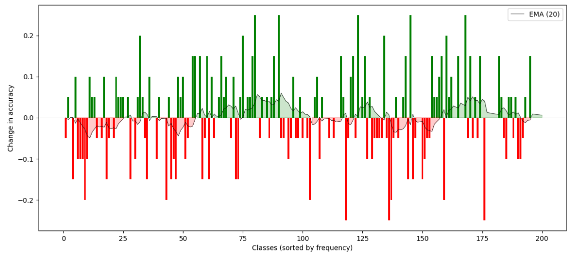

Figure 5. Change in accuracy from threshold of 0.8 to iteratively decaying threshold across classes. Green bars indicate better performance and red bars indicate worse performance for the corresponding class

Our model used the update rule detailed in 2.2, with the threshold being decayed by a small amount every iteration based on the model's confidence. The range of the threshold for the experiment was 0.8-0.95.

This plot highlights how changing the threshold affects classes based on their frequency. Underrepresented have an increased accuracy when the global threshold is iteratively decayed. However, this comes with the trade-off in accuracy across overrepresented classes, similar to linearly decaying global threshold. As compared to linearly decaying thresholds, a fewer number of overrepresented classes have a drop in accuracy. The performance improvement over underrepresented classes improve with the iterative decay because the model starts with a high threshold and therefore prevents the pseudo-labels from overrepresented classes dominating the training process. As the threshold decays, less confident pseudo-labels are incorporated allowing diversity in training data. This includes pseudo-labels from underrepresented classes that were previously excluded due to their lower confidence. On the other hand, by preventing the domination of overrepresented classes, the accuracy over this decreases. This trade-off allows the model to better focus on underrepresented classes.

4.2.3 Accuracy for global thresholds

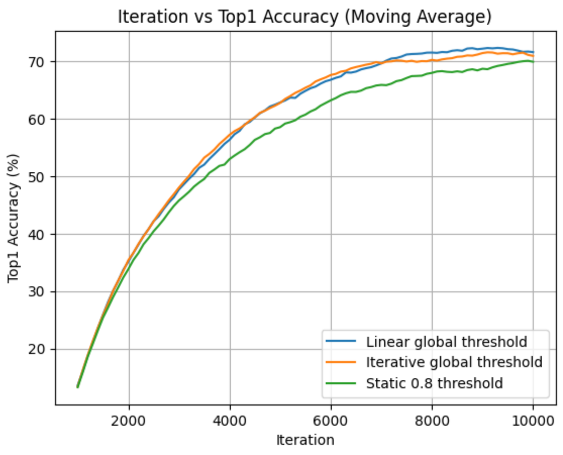

Figure 6. Top 1% accuracy plotted every 200 iterations for the training set for different thresholds

With the new adaptive thresholds, from the plot 6, the top 1% accuracy for the training set over iterations improved compared to the static global threshold of 0.8. The static threshold has a flatter curve and shows a slower initial improvement, reaching a lower accuracy compared to the other two methods. However, there was not a significant difference in test accuracy for the linearly decaying threshold as compared to iteratively decaying threshold.

Method	Top1 Acc (%)	Top5 Acc (%)
Static (0.8)	40.45	71.65
Linear	41.93	72.22
Iterative	40.55	72.30

Table 2. Comparison of methods for different thresholds.

For the test set, the data is detailed in Table 2. Both the adaptive thresholds have a higher accuracy (Top 1% and Top 5%) as compared to a static threshold of 0.8. Though iterative global threshold has a lower top 1% accuracy as compared to linear global threshold, it has a higher top 5% accuracy.

4.3. Adaptive Class based thresholds

4.3.1 Static class based threshold

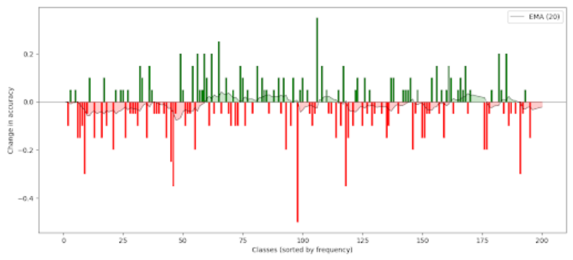

Figure 7

Figure 7 shows the comparative analysis of a static threshold of 0.8 and the change in accuracy by adding in static class based thresholds. As mentioned in 2.2, the static class based thresholds are proportional to frequency, which means overrepresented classes have a higher frequency.

Since there is more labeled images related to the overrepresented classes available for training, intuitively, the model would be capable of capturing a more accurate representation of these classes during training, thereby enabling it to make more confident predictions on them. Hence, for pseudo-labeling unlabeled data, we assign a higher threshold for overrepresented classes. In comparison, for underrepresented classes, we assign a lower threshold since we assume that the model would be less confident in dealing with these classes due to the smaller amount of labeled data related to these classes available to the model. Since the thresholds only allow confident predictions to be used for pseudo-labeling, we ensure that underrepresented classes, which would intuitively have lower confidence, would be provided a more relaxed threshold to ensure that they are provided a fair chance at being represented as well in the model. This would help us to reduce the bias the model might have for pseudo-labeling overrepresented classes purely due to their higher frequency.

In this case, classes in the middle (with not too many labeled data points, seem to be performing better. In comparison, it seems that setting a static higher threshold for overrepresented classes while giving a lower threshold for underrepresented classes resulted in a loss in overall accuracy for overrepresented classes. However, there does seem to be some improvement in the model's accuracy when dealing with underrepresented classes.

4.3.2 Iteratively decaying class based threshold

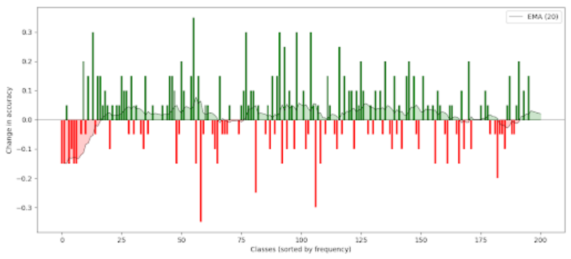

Figure 8

Figure 8 shows the change in accuracy per class after using iteratively decaying class-specific thresholds instead of a static threshold of 0.8. The iterative class-specific thresholds are updated at each iteration based on the confidence of the class. It can be seen that this approach improves accuracy on underrepresented classes in the dataset.

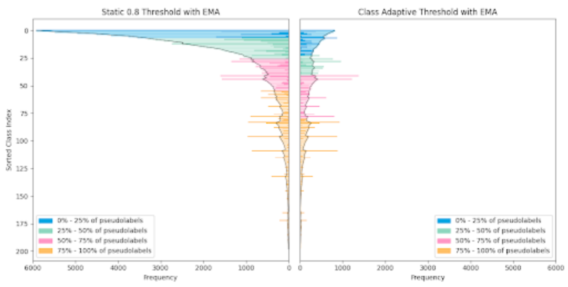

Figure 9. Figure 9. Number of pseudo-labels generated for each class for static 0.8 threshold (Left) and class-specific threshold (Right), color coded by percentile

The above figure illustrates the number of pseudo-labels generated for each class under a static threshold of 0.8 (left) and class-specific threshold (right). The bars are color-coded by percentile to show the distribution of pseudo-labels across the classes. As shown on the left, there is a skewed distribution of pseudo-labels, where overrepresented classes tend to dominate the labeling process. In contrast, on the right side, the class-specific thresholds ensure a more balanced distribution of pseudo-labels. This shift explains the change in accuracy displayed in Figure 8. Our class-specific thresholding approach successfully addresses the class imbalance issue.

4.3.3 Test accuracy for class based thresholds

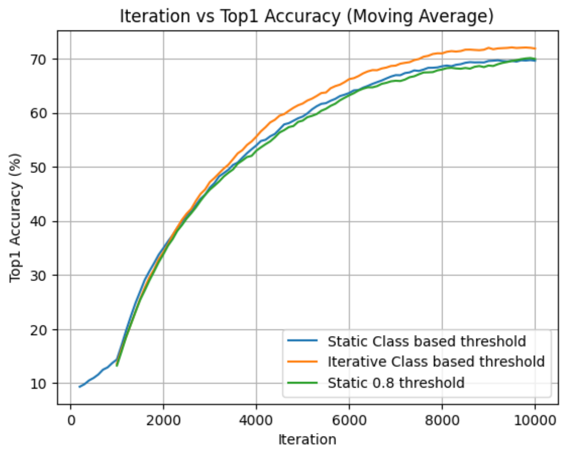

Figure 10

The training accuracy per iteration after adding class-specific thresholds improves as compared to a static threshold of 0.8. The learning curve indicates that the model continues to learn features with class-specific thresholds, when it doesn't with static 0.8 thresholds which has a flatter curve. It is expected that this improvement in accuracy comes from the model learning better over underrepresented classes.

For the test set, the top 5% accuracy for class-specific thresholds is higher than static threshold of 0.8, with iterative class based thresholds resulting in a 73.1% accuracy. Also, in case of top 1% accuracy, iterative class-specific has a 1% increase in accuracy.

Method	**Top1 Acc (%)**	**Top5 Acc (%)**
Static (0.8)	40.45	71.65
Naive Class Based	39.45	72.12
Iterative Class Based	41.77	73.10

Table 3. Comparison of Top1 and Top5 accuracies for different methods.

4.4. Combining Class based threshold and Global Thresholds

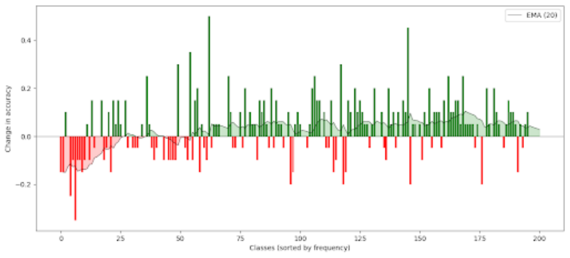

Figure 11

The harmonic mean of the class-specific thresholds with the global threshold is set as the final threshold value per class. In this case, it is surprising to note that harmonic mean has comparable results to iterative class based thresholds 8. Global thresholds would have only a small impact on the final thresholds at every iteration, and class-specific thresholds would hold a higher priority. This could lead to future work with finding a different approach to combining global and class-specific thresholds.

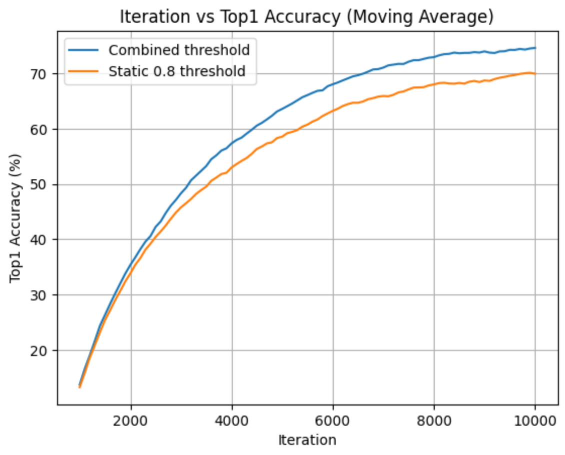

Figure 12

Figure 12 shows that there is significant improvement in train accuracy for combined threshold with harmonic mean, as compared to the initial static threshold of 0.8. Train accuracy for the combined threshold is 76.6% whereas for static 0.8 threshold is 68.62%.

Looking a test accuracy, the combined threshold model does significantly better than the static threshold of 0.8. This improvement can similarly be attributed to the improvement in accuracy over underrepresented classes.

Method	**Top1 Acc (%)**	**Top5 Acc (%)**
Static (0.8)	40.45	71.65
Combined threshold	43.70	74.45

Table 4. Comparison of Top1 and Top5 accuracies for different methods.

4.5. Using a larger pre-trained model: RESNET-50

For running pseudolabeling on semi-fungi, a ResNet model [9] was used. Larger models are generally expected to perform better, though they would require significant compute. Experiments were run with ResNet50 and ResNet18, to compare accuracy on a small number of iterations. Due to limited resources, the model used for experiments is ResNet18.

For 1000 iterations, ResNet50 took 21m 25s, whereas ResNet18 took 11m 38s. The Top 1 accuracy for ResNet50 was 10.32% which could be compared to ResNet18 of 8.40%. The Top 5 accuracy for ResNet50 was 24.05% which could be compared to ResNet18 of 21.18%.

Looking at the accuracy results for ResNet18, it was noted that Adaptive Class based model does slightly better than the Combined Threshold model. So, for the comparision to ResNet50, the Adaptive Class based model was chosen. ResNet50 does significantly better due to being a better based model. The results from ResNet50 as compared to ResNet18 with Adaptive Class based thresholds are mentioned in 5

Iterative Class Based	**Top1 Acc (%)**	**Top5 Acc (%)**
ResNet18	41.77	73.10
ResNet50	47.55	80.17

Table 5. Comparison of Top1 and Top5 accuracies for different methods.

5. Conclusion

Fine-grained image classification tasks require the model to capture subtle distinctions in images. Pseudo-labeling was used on the Semi-Fungi dataset for image classification. Given the long-tailed nature of the distribution, the naive model (static threshold) does better on certain classes with higher samples in the dataset.

Our project aimed to address this imbalance by using adaptive thresholds to improve the performance over underrepresented classes, while maintaining overall accuracy. Our approach began with testing a naive approach of adapting thresholds over training and over classes, which delivered promising results.

Building on this foundation, we proposed an iterative method to update the global and class-specific thresholds based on the confidence of the model. Our model significantly improved the performance of underrepresented classes, with a small increase in overall accuracy. However, this improvement came at the cost of reduced performance on overrepresented classes. Future work will focus on exploring alternative strategies for combining global and class-specific thresholds to achieve a better balance across all classes.

References

[1] Eric Arazo, Diego Ortego, Paul Albert, Noel E. O'Connor, and Kevin McGuinness. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. 2

[2] Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles, 2014. 2

[3] beejisbrigit and Yin Cui. 2018 fgcvx fungi classification challenge. `https://kaggle.com/competitions/fungi-challenge-fgvc-2018`, 2018. Kaggle. 1

[4] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning, 2019. 2

[5] David Berthelot, Rebecca Roelofs, Kihyuk Sohn, Nicholas Carlini, and Alex Kurakin. Adamatch: A unified approach to semi-supervised learning and domain adaptation, 2022. 2

[6] Paola Cascante-Bonilla, Fuwen Tan, Yanjun Qi, and Vicente Ordonez. Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning, 2020. 2

[7] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006. 1

[8] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations, 2018. 2

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016. 7

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

[11] Dong-Hyun Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 2013. 1, 2

[12] G. J. McLachlan. Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. *Journal of the American Statistical Association*, 70(350):365–369, 1975. 2

[13] Sylvestre-Alvise Rebuffi, Sebastien Ehrhardt, Kai Han, Andrea Vedaldi, and Andrew Zisserman. Semi-supervised learning with scarce annotations, 2020. 2

[14] Mehdi Sajjadi, Mehran Javanmardi, and Tolga Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning, 2016. 2

[15] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. In *Advances in Neural Information Processing Systems*, pages 596–608. Curran Associates, Inc., 2020. 2

[16] Yidong Wang, Hao Chen, Qiang Heng, Wenxin Hou, Yue Fan, Zhen Wu, Jindong Wang, Marios Savvides, Takahiro Shinozaki, Bhiksha Raj, Bernt Schiele, and Xing Xie. Freematch: Self-adaptive thresholding for semi-supervised learning, 2023. 2, 3

[17] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training, 2020. 2

[18] Yi Xu, Lei Shang, Jinxing Ye, Qi Qian, Yu-Feng Li, Baigui Sun, Hao Li, and Rong Jin. Dash: Semi-supervised learning with dynamic thresholding, 2021. 2

[19] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning, 2019. 2

[20] Bowen Zhang, Yidong Wang, Wenxin Hou, Hao Wu, Jindong Wang, Manabu Okumura, and Takahiro Shinozaki. Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling, 2022. 2