INF473V: Semi supervised classification challenge

Antoine Bonnet, Ecole polytechnique, antoine.bonnet.x21@polytechnique.edu
Alae Ahmad, Ecole polytechnique, alae.ahmad@polytechnique.edu

Abstract

In this report, we discuss different variants of pseudo-labeling methods that we have implemented to address the Semi-supervised Challenge proposed by Vicky Kaleigton, Nicolas Dufour and Pascal Vanier during the INF473V class at Ecole Polytechnique. We also attempted to use a StableDiffusion Pipeline to do data augmentation.

1. Introduction

In recent years, Deep neural networks have achieved outstanding results in supervised settings. For example, the ImageNet challenge was closed in 2017 because models managed to achieve error rates lower than 3%. If deep neural networks can achieve very strong performances through supervised learning, labeled data is difficult to acquire in large quantity. It has to be done by humans and it can be very costly, especially when it has to be performed by an expert like in medical applications.

The final challenge of the INF473V course is a semi-supervised classification problem. We are given a dataset of 300 000 images that belong to 48 distinct classes. But only 720 images are labeled (15 for each class). The goal is to train a model that has the highest accuracy on a given sample of the unlabeled dataset. The images that were provided are artificial images generated by an AI and we are allowed to learn from unlabeled images even though we don't know the corresponding labels.

The main technique to address this problem is pseudo-labeling. It was first described in [1]. The first training consists in training the model on the labeled data. The second step of training consists in making the model label the unlabeled data and using the produced label to further train the model.

2. Methodology

2.1. Training set and validation set

First all images are normalized with the mean and the standard deviation of the ImageNet dataset. Then, in order to keep track of the performances of our models, we created a set of labeled images on which we never train.

To do so, we split the labeled dataset into two distinct datasets : a training dataset and a validation dataset.

We fixed the seed to ensure that the validation dataset stayed the same for each experiment to be able to reuse previous models after saving them.

2.2. Data Augmentation

We created an augmented dataset that contains the same images as the train dataset but where the images are augmented. We tried different data augmentation strategies including rotations, translations, colorJittering but also MixUp and RandAugment.

Left: an image of the raw dataset. Middle: the same image in the train dataset. Right: the same image in the augmented dataset with soft augmentation.

We also tried a different approach : we wanted to expand our limited train dataset by generating images that correspond to the class names using a text to image generative AI. We used the StableDifusionPipeline from HuggingFace.co which has the advantage to be free. You may find the augmented images and the models that we have trained in the following Kaggle dataset : https://www.kaggle.com/datasets/antoinebonnet2001/generated-images

2.3. Pseudo-labeling

To address this semi-supervised problem, we implemented different variants of Pseudo-labeling methods that

are inspired from the initial pseudo-labeling method [1] and the Fixmatch paper [2].

In Lee's article, the loss used for backpropagation was computed in the following manner, see equation (15) in [1]:

$$\text{loss} = \text{labeled loss} + \alpha(t) \cdot \text{unlabeled loss} \tag{1}$$

Where $\alpha(t)$ is a parameter which stays at 0 for a predetermined number of epochs (initial training on labeled data) and then increases over the epochs. In this first paper, it is interesting to notice that there is no test about how confident the model is for the generated label.

To describe the threshold test for pseudo-labeling we will be using the FixMatch paper even though we didn't manage to implement the FixMatch algorithm. For a L-class classification problem, let $\mathcal{X} = \{(x_b, p_b) : b \in (1, ..., B)\}$ be a batch of B labeled examples where x_b is the image and p_b the corresponding label. Let $\mathcal{U} = \{u_b : b \in (1, ..., \mu B)\}$ a batch of unlabeled images where μ is the relative size. By using a softmax, The models that we will consider will output a distribution of probability $p_m(y|x)$. For an unlabeled image u_b we denote $q_b = p_m(y|u_b)$ the output of the model and $\hat{q}_b$ the hard label associated. The unlabeled images u_b used for training will only be the ones where $\max_y p(y|u_b) > \tau$ where τ is the threshold parameter. We denote $H(p, q)$ the cross entropy between two probability distribution p and q. The loss is computed as follows:

$$l_s = \frac{1}{B} \sum_{b=1}^{B} H(p_b, p_m(y|x_b)) \tag{2}$$

$$l_u = \frac{1}{\mu B} \sum_{b=1}^{\mu B} 1(max(q_b) \geq \tau) H(\hat{q}_b, p_m(y|u_b)) \tag{3}$$

$$\text{loss} = l_s + \lambda_u \cdot l_u \tag{4}$$

We will choose like in [2] that $\lambda_u = 1$. From this theory we derived three training procedures.

2.4. Training on labeled and unlabeled data at the same time : Classical Pseudo-Labeling

This method is exactly the one described before. We train the model on the unlabeled data selecting only the unlabeled data that passes the threshold test. And every μ batches of unlabeled data, we train the model on one batch of labeled data.

2.5. Permanent labeling : label the unlabeled data only once

With the previous method, different labels can be assigned to a sample of unlabeled data over the course of training, hence we decided to experiment what happens if we label the unlabeled data once and for all, adding them to the training dataset for good with their first assigned label.

2.6. Local labeling: Train on labeled and unlabeled separately

This time we decided to separate the second phase of training in two phases. First, we pseudo label the unlabeled data and we store the addresses of the selected images that passed the threshold test. Then, we train the model on that selected unlabeled data. After this step, we retrain the model on the labeled data and we repeat these two steps.

2.7. Choice of model and hyperparameters for experiments

For the choice of the model, we tried to finetune a Resnet50 architecture. We also tried to finetune a VIT-ForImageClassification from hugging.co. For each of these two models we tried to compare what happens if we freeze the computation of the gradient on every layers except the classifier layer (which is the principle of transfer learning). It that is the case we say that Frozen = True, else we say that Frozen = False.

For the optimizer, by testing only on the train dataset, we've concluded that SGD with momentum = 0.9 achieved overall better performance than Adamw like in [2]. We used a learning rate $\eta = 0.001$ without any scheduler. We also tried $\eta = 0.01$: the speed of convergence was faster but it yielded less accurate results. We used batch size = 128 for Resnet50 and batch size = 64 for the transformer due to GPU memory constraints. We used $\mu = 7, \lambda_u = 1$ for the classical pseudo-labeling method like in [2].

3. Results

3.1. Data Augmentation

We have found that training on the augmented dataset wasn't very different than training on the train dataset. We didn't see very different results on the accuracy. Very often, the validation accuracy decreased when we trained on the augmented dataset (no matter the augmentation).

We tried to use the images generated with the StableDiffusionPipeline as additional training data on a model already trained on the train dataset. We started with a trained model with a validation accuracy of 40% and the accuracy plunged from 40% to less than 2% in a few epochs. It didn't work at all. This can be explained because we don't know the exact generative model and the exact prompt that was used the generate our images. We've actually noticed that the AI-generated images looked too realistic compared with the ones in the dataset.

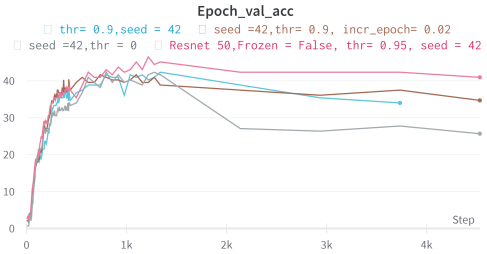
Overall, our data augmentation strategies were unsuccessful.

In the following sections, the ablation study was conducted only on 30% of the unlabelled dataset and 4 epochs

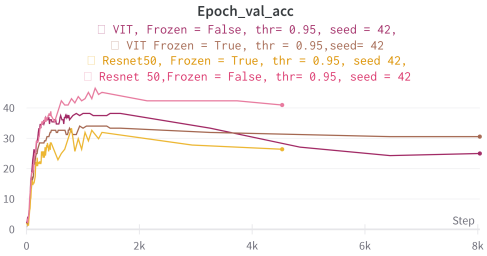
were run on this 30% of the unlabeled dataset. The seed was fixed for every experiment.

3.2. Classical Pseudo labeling

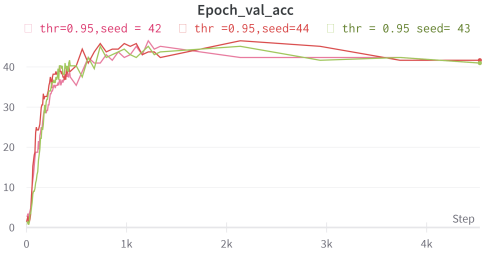
First, we compared the validation accuracy of Resnet50 with frozen = False, with different thresholds: 0, 0.9, 0.95 and one that starts at 0.9 and which increases by 0.02 every epoch on the unlabeled dataset. To produce the graphics, we used the wandb package and fixed the seed at the value 42. We then logged the validation accuracy into WandB.



We see that the the best setting is when thr = 0.95. Then, we compared different backbones.



We have found that Resnet50 is more efficient than our ViT. Furthermore, Resnet50 is more efficient with Frozen = False. To our knowledge, we were the first group to discover that and we were very surprised because it seems somehow opposite to the principles of transfer learning that we learned in the lectures. Thus, we have found that the best settings are Resnet50 with Frozen = False and threshold = 0.95.



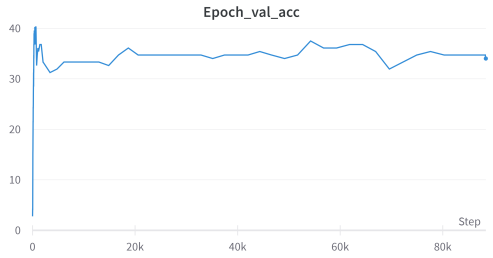
In this figure we can see that since the validation set is quite small the validation accuracy can differ by few percents between different seeds.

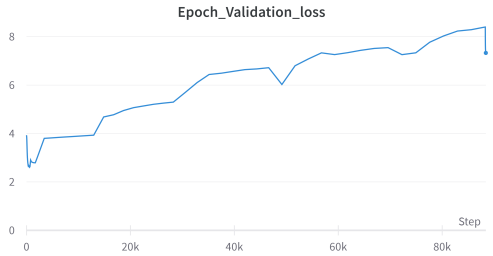
This method achieved decent results : up to 47% on the leaderboard when taking the best state.

3.3. Permanent labeling

We initially thought that training on more images per epoch would be beneficial, which is why we tried ”permanent labeling” : once an image is labelled, we add it to the training dataset forever.

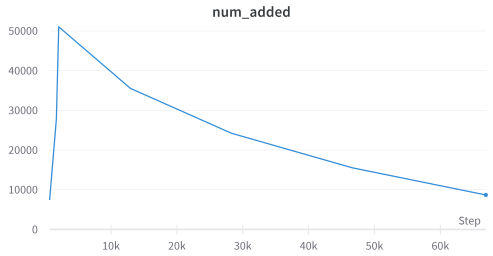
However, it turns out that this method was even worse than training only on the train dataset. We only did a 20-hour run with a thr = 0.95 and Resnet 50 Frozen = False on seed 42.





As we can see, the validation accuracy remains stable or even decreases with each epoch, while the validation loss skyrockets. What's curious is that the two don't seem to be correlated at all.

A plausible explanation is that we initially label a lot of images with a model that is still not accurate enough, and once they're (falsely) labeled, these images continue to haunt the model until the very end. On top of that, we add less and less new data to the dataset per epoch, even though they're supposed to be more accurately labeled after each epoch...



This method didn't really work, but it lead us to one conclusion : Quality over Quantity.

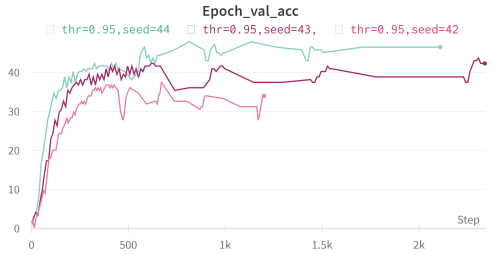
3.4. Local labeling

Here are the results we obtained training the model with Resnet50 :

	Seed = 42	Seed = 43	Seed = 44
thr=0.95, Frozen=True	Acc=0.370, Loss=2.67	Acc=0.472, Loss=2.54	Acc=0.491, Loss=2.31
thr=0.95, Frozen=False	Acc=0.284, Loss=2.90	Acc=0.347, Loss=2.80	Acc=0.333, Loss=2.85
thr=0.90+0.02*e, Frozen = False	Acc=0.415, Loss=2.69	Acc=0.447, Loss=2.64	Acc=0.468, Loss=2.52

It seems that a higher threshold from the beginning is better, even though we add significantly less images to the training dataset at each epoch.

Let's take a look at the different seeds for the best threshold value (thr= 0.95) with the Resnet50 architecture.



In this figure we can see that since the validation set is quite small the validation accuracy can differ by few percents between different seeds.

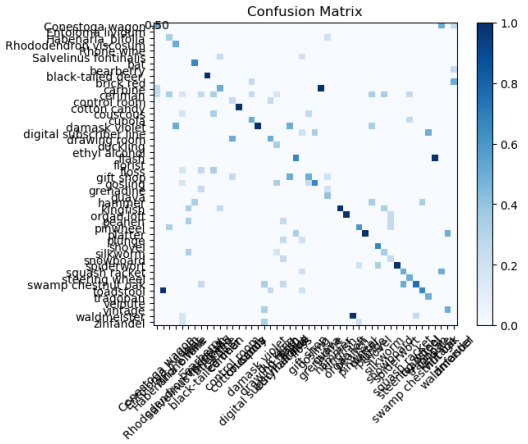
We can clearly see that the major part of the training is actually during the initial training phase with the labeled data.

We noticed that there were huge differences between the different seeds. The seed 42 was catastrophic, resulting in a 35% accuracy, which is strictly worse than the initial training alone. With the other seeds (43 and 44), the accuracy increases overall, and sometimes even increases before retraining on the initial labeled data. We have managed to reach a peak of 50% validation accuracy with seed 44.

We also tried other backbones, such as VIT (which got stuck at 40%) , but Resnet50 turned out to be the best. We found again that unfreezing the Resnet model and the VIT leads to much better results, which is probably due to the fact that the images of this dataset are farther away than those of ImageNet.

3.5. Confusion matrix

In order to determine on which classes the model performs well, we built a confusion matrix to visualize the predictions on the validation set.



However, because of the small length of the validation dataset, it is hard to tell whether these differences between classes are only due to chance.

4. Conclusion

With the various simple pseudo-labeling methods presented here, we are forced to admit that the results didn't live up to our expectations. Pseudo-labeling barely improved the performance of the model compared to just training more on the labeled dataset, and classical methods such as data augmentations turned out not very useful. More sophisticated methods, like the FixMatch algorithm, seem to be needed to really make a difference.

However, the amount of experiences with different hyperparameters and backbones, made us realize how crucial these choices are, even when they seem insignificant (using different seeds), and even when they seemed worse (Frozen = False turned out to be way better than Frozen = True !).

We probably could've tried to use neural networks that are already pre-trained on (image, text) pairs, like CLIP from OpenAI.

References

[1] D.-H. Lee. Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks. *In ICML Workshop on Challenges in Representation Learning, 2013*, 2013. 1, 2

[2] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *CoRR*, abs/2001.07685, 2020. 2