Universität des Saarlandes
Prof. Dr. Dietrich Klakow
Lehrstuhl für Signalverarbeitung
NNTI Winter Term 2021/2022

Computer vision project

Semi-supervised Image Classification

Deadline: 04.03.22 23:59

1 Introduction

Deep neural networks have been successfully applied to several applications in computer vision. Their success can be attributed to their scalability and ability to generalize on several tasks. One of the critical drawbacks of neural networks, is that they have to be trained on large datasets for them to achieve any reasonable performance. This also adds to the cost of requiring to label, store and curate large datasets for training them. As an example, the largest datasets today have samples in the order of millions of images and memory requirements in terabytes. In a few domains like medical imaging or astrophysics, labeled data can be notoriously difficult to obtain. One of the ways we can train deep networks without having a lot of labeled data is semi-supervised learning (SSL). The idea behind SSL is to develop training strategies that learn from labeled data as well as unlabeled data.

In this project we are going to implement common techniques for SSL and challenge you to beat a benchmark result for semi-supervised image classification. The first task is to implement a standard SSL algorithm for image classification. The second task asks you to implement a more recent technique while the third task introduces a simple but state of the art method which you will try to improve upon. We are going to denote our labeled samples with $\{X_i^{(l)}, y_i^{(l)}\}_{i=1}^m$ and our unlabeled samples with $\{X_i^{(u)}\}_{i=1}^n$ and in general we assume that number of labeled samples are far less compared to number of unlabeled samples ($m \ll n$). In practice, to create such a setting for our experiments we do the following for a particular dataset $\{X_i^{(l)}, y_i^{(l)}\}_{i=1}^z \in \mathcal{X}$.

Algorithm 1 Create labeled and unlabeled data

```
1: Start with a standard labeled dataset $\{X_i^{(l)}, y_i^{(l)}\}_{i=1}^z \in \mathcal{X}$.
2: Randomly select a subset of $m$ samples from $\mathcal{X}$ with labels for creating $\{X_i^{(l)}, y_i^{(l)}\}_{i=1}^m$.
3: Discard the labels for the rest of the dataset for creating $\{X_i^{(u)}\}_{i=1}^n$.
Ensure: $(m + n = z)$
```

2 Datasets

In this project we are going to use CIFAR-10 and CIFAR-100 datasets [4] for our experiments, since they are relatively small and can be used with moderate compute resources. Each dataset has 60000 images with 50000 training images and 10000 test images. CIFAR-10 has 10 classes which are given in figure 1. CIFAR-10 has 5000 training images per class and 1000 test images per class. CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes containing

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food	containers bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Table 1: CIFAR-100 dataset classes. The 100 classes are grouped into 20 "superclasses" of broad categories

600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). Table 1 gives the list of classes in the CIFAR-100. The purpose of using both datasets is to choose one dataset which has more samples per class (CIFAR-10) and another dataset which has less samples per class (CIFAR-100).

In this project we are going to set the number of label data we are going to use for each task. *For CIFAR-10 you are going to report your results by only using 250 and 4000 labeled samples from the dataset for each task. That also means you have 25 and 400 labeled samples per class respectively. For CIFAR-100 you are going to report your results using 2500 and 10000 labeled samples from the dataset for each task. That means in CIFAR-100 you have 25 and 100 labeled samples per class respectively.* We now go over the different tasks you will be asked to submit

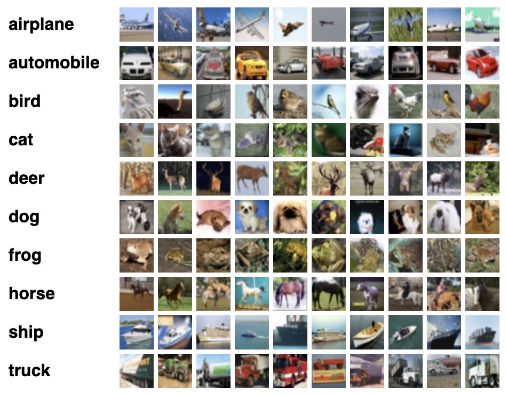

Figure 1: CIFAR10 classes and sample images

the code for.

3 Task 1: Pseudo-Labeling (8 points)

Pseudo-labeling [5] is a contemporary technique in SSL which assigns labels to the unlabeled data $\{X_i^{(u)}\}_{i=1}^{n}$ using the predictions of the network trained on the labeled data $\{X_i^{(l)}, y_i^{(l)}\}_{i=1}^{m}$. The idea is quite simple, first we train our network using our labeled data and then using our trained network, we predict the class probabilities of our unlabeled data. If the probability of a class is greater than a user defined threshold value for a sample, we assign the sample that particular class and use it for training. We repeat this procedure for every epoch of training.

Algorithm 2 Pseudo-labeling algorithm

```
Require: Labeled samples $\mathcal{L} = \{X_i^{(l)}, y_i^{(l)}\}_{i=1}^{m}$
Require: Unlabeled samples $\mathcal{U} = \{X_i^{(u)}\}_{i=1}^{n}$
Require: Confidence threshold $\tau$
1: $X_t = \{\}$                                              ▷ Pseudo-labeled set
2: for each training epoch do
3:     Train model $M$ on $\mathcal{L} + X_t$
4:     Find class probabilities $C = M(\mathcal{U})$ for unlabeled data $\mathcal{U}$
5:     $X_t = \{\}$
6:     for $u_i$ in $\mathcal{U}$ do
7:         if $max(c_i) > \tau$ then
8:             $X_t = X_t \bigcup (u_i, pseudoLabel(u_i))$    ▷ If the probability of a class is greater than
   threshold, then add the sample to pseudo-labeled set
```

Implement the pseudo labeling algorithm 2 and report your error on the test set of CIFAR-10/100 for the different number of labeled samples mentioned above. Choose different thresholds $\tau = \{0.95, 0.75, 0.6\}$ and report your error for each value. You may choose to modify algorithm 2 slightly to get better results. For example by using a learning rate scheduler or training by using only labeled data for the first few epochs. But the overall idea should be as described.

3.1 Other instructions for task 1

- We have given a dataloader that implements algorithm 1 and creates torch dataset of CIFAR-10/100 of labeled, unlabeled and test samples. You are to use this dataloader only and not create one of your own for this task. How to use the dataloader is given in main.py of the task directory. You are free to modify main.py but not dataloader.py.
- Create a table in your report that gives the error rates on different number of labeled samples for CIFAR-10 (250 and 400) as well as CIFAR-100 (2500 and 10000). You can see table 2 in [7] as an example of how we expect you to report your results.
- Try different confidence threshold values $\tau = \{0.95, 0.75, 0.6\}$.
- In the README.md file of the task folder, give detailed instructions on how to reproduce your results with correct arguments.
- You are only allowed to use WideResNet [8] as the base model for this task. The depth and width of the model is your choice. We have given the implementation of WideResNet along with the code.

4 Task 2: Virtual Adversarial Training (10 points)

Another simple SSL method is virtual adversarial training (VAT)[6]. The idea behind VAT stems from adversarial examples. We can perturb an image with noise so small that it is invisible to the eye and yet the neural network may produce a different classification for the perturbed image than the original image. In VAT we deliberately add noise to an image in such a way that the network misclassifies the image. In adversarial training we add adversarial noise to the images and continue training our network. The network will be robust to test images perturbed with adversarial noise.

With VAT [6] which is a similar idea to adversarial training, you can train your network with sparsely labeled data (e.g. SSL). In VAT we try to find a perturbation (r) that maximizes the KL divergence between the original image and the adversarial image. We then train the network to predict the same label for original image and perturbed image. The network is penalized if it predicts a different label for the original and perturbed image.

Algorithm 3 Virtual Adversarial Loss

```
Require: $\xi$: hyperparameter in [1, 10]
Require: $\epsilon$: hyperparameter in [0,1]
Require: vatIter: hyperparameter for number of VAT iterations
 1: procedure VATLoss(model, $\mathcal{U} = \{X_i^{(u)}\}_{i=1}^n$)
 2:     Sample a random tensor r from a gaussian distribution
 3:     $r$ = L2Norm(r)
 4:     predictions = model($\mathcal{U}$)
 5:     for number of vatIter do
 6:         advExamples = $\mathcal{U} + \xi r$
 7:         advPredictions = model(advExamples)    ▷ Do not forget to apply softmax to both predictions
 8:         advDistance = KLDivergence(predictions, advPred)
 9:         Calculate the adversarial perturbation r_adv by taking the gradient on advDistance (d)
10:     r_adv = L2Norm(d) * $\epsilon$
11:     advPredictions = model($\mathcal{U}$ + r_adv)
12:     loss = KLDivergence(predictions, advPredictions)
13:     return loss
```

Algorithm 4 Virtual Adversarial Training Algorithm

```
Require: Labeled samples $\mathcal{L} = \{X_i^{(l)}, y_i^{(l)}\}_{i=1}^m$
Require: Unlabeled samples $\mathcal{U} = \{X_i^{(u)}\}_{i=1}^n$
1: for each training epoch do
2:     Sample labeled batch $(x^l, y)$
3:     Sample unlabeled batch $x^u$
4:     vaLoss = VATLoss(model, x)
5:     predictions = model($x^l$)
6:     classifcationLoss = (predictions, $y$)
7:     loss = classificationLoss + $\lambda$ vaLoss
8:     update model parameters
```

The key idea is to find a perturbation the maximizes the KL divergence between the adversarial image and original image. Algorithm 3 describes how to find r the maximizes the KLD between the adversarial image and original image. Then it calculates the VAT loss between perturbed image and original image

For this exercise you need to implement Algorithm 4 and visualize the adversarial examples. Read the paper for VAT [6] and implement their algorithm as described in §3.2 of the paper. Report the error you make on the test set of CIFAR-10/100 similar to Task 1.

4.1 Other instructions for task 2

- We have given a dataloader that implements algorithm 1 and creates torch dataset of CIFAR-10/100 of labeled, unlabeled and test samples. You are to use this dataloader only and not create one of your own for this task. How to use the dataloader is given in main.py of the task directory. You are free to modify main.py but not dataloader.py.
- Create a table in your report that gives the error rates on different number of labeled samples for CIFAR-10 (250 and 400) as well as CIFAR-100 (2500 and 10000). You can see table 2 in [7] as an example of how we expect you to report your results.
- Give some examples of the adversarial images that get created using the adversarial perturbation in the report.
- In the README.md file of the task folder, give detailed instructions on how to reproduce your results with correct arguments.
- You are only allowed to use WideResNet [8] as the base model for this task. The depth and width of the model is your choice. We have given the implementation of WideResNet along with the code.
- Some implementations calculate the VAT loss for both the labeled as well as unlabeled images. You can choose to do that or use just the unlabeled images.

5 Task 3: Challenge Task (12 points)

In this task, you are free to implement your own idea for SSL. **Note that the idea you propose should be your own and not any reimplementation of existing work**. You have to keep the number of labeled data same as in the previous two settings. But you may use a model different to wideresnet which has less then 30M parameters. In this task you are supposed to try to improve upon a state of the art algorithm for SSL, fixmatch [7] so go through the paper atleast once and try to understand its details. Although fixmatch[7] gives state of the art results, it is relatively easy to understand. We describe the algorithm briefly.

In Fixmatch, we train a supervised model on our labeled images with cross-entropy loss. For each unlabeled image, weak augmentation and strong augmentations are applied to get two images. The weakly augmented image is passed to our model and we get prediction over classes. The probability for the most confident class is compared to a threshold. If it is above the threshold, then we take that class as the ground label i.e. pseudo-label. Then, the strongly augmented image is passed through our model to get a prediction over classes. This probability distribution is compared to ground truth pseudo-label using cross-entropy loss. Both the losses are combined and the model is tuned.

Other methods like alphamatch [2], flexmatch [9] and simPLE [3] improve upon fixmatch [7] so you can look at their ideas. It is completely alright if you aren't able to get the desired results that improve upon fixmatch. We care more about the ingenuity of your ideas and the analysis of how and why it works!!

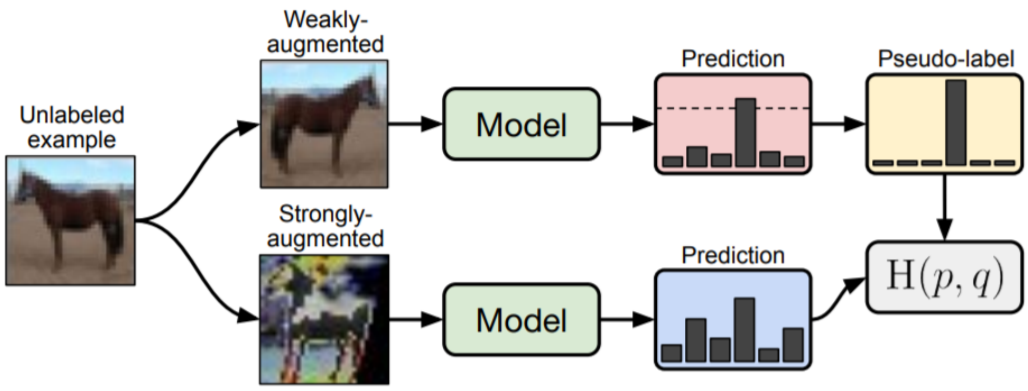

Figure 2: Diagram of FixMatch taken from the paper [7]. A weakly-augmented image (top) is fed into the model to obtain predictions (red box). When the model assigns a probability to any class which is above a threshold (dotted line), the prediction is converted to a one-hot pseudo-label. Then, we compute the model's prediction for a strong augmentation of the same image (bottom). The model is trained to make its prediction on the strongly-augmented version match the pseudo-label via a cross-entropy loss.

5.1 Other instructions for task 3

- Unlike task 1 and task 2, here you are free to define your own code and parameters for training as SSL model. You can design a dataloader with your own augmentation strategies (e.g. RandAugment [1])
- You have to supply a test.py file that can be used to test any CIFAR like dataset.
- Use only open source models in your implementations. Note that it should only be pretrained on ImageNet or randomly initialized. Limit is 30M parameters.
- No commercial libraries are allowed.

6 Report

You have to submit your project report in NeurIPS conference paper format for which the latex template is given here. You can also use other formats for writing your report but ultimately you should submit a pdf file. The page limit is **6 pages**, excluding references and this is a hard limit. You must use the **camera ready** version for your submission. The abstract should be short (4-5 sentences) and the rest of the report should focus on your implementation details and results. Typically, you should follow the format of 1-1.5 pages for introduction and related work, $\sim 2-2.5$ pages for describing your implementations and $\sim 2-2.5$ pages for discussion of your results. If you need additional space, you may use an appendix after your references but your appendix should contain only figures, result tables and graphs. Your appendix **must not** contain any substantial text.

7 Implementation instructions

- In case you face any difficulties with the project, please raise a question on piazza. Alternatively you can get in touch with the tutor in charge (Kaustubh) using MS-Teams.
- Plagiarism will be penalized and can eventually lead to disqualification from the project and the course. Most importantly, we will check for plagiarism within groups. If we see

any clear indication of plagiarism among groups, both the groups will be awarded 0 for the whole project. Discussion with groups is allowed (only in terms of concepts but not directly with code). Cite all your sources for code and report.

- **Tip:** First finish every task for CIFAR-10/100 using the highest number of labeled samples (4000 and 10000 respectively). Then aim for reducing the number of samples.
- You may notice that we use a python iterator to loop over the dataloader unlike most other pytorch training code. This is because we loop over two datasets of unequal sizes and sample a batch from each dataset in the next iteration. You may use a torch batch sampler for this in task 3.
- For each task we have asked you to implement a test.py for testing your models. Implement that function and describe in the README all the models that you save.
- You have to create a README file along with the submission which explains how to run your code and how to reproduce each result that you have presented in your report.
- Use tensorboard to visualize your training and test curves. Also report the error on the test set of CIFAR-10/100.
- Do not submit the dataset along with your submission, instead include a data path argument in your python files.
- As a base model, you must use only wideResnet [8] for the first two tasks[1]. It must be either pretrained on ImageNet or randomly initialized. You must not use any other open source pretrained model.
- For the third task you may use a model either pretrained on ImageNet or randomly initialized but with less than **30M parameters**. These limits are enforced in order to keep training time and compute resources reasonable.
- Each task is in a separate folder of the main directory. You must save all your trained models and submit them.
- A common mistake is to use a model that was meant for an input size of 224×224 (ImageNet) and modify it for an input size of 32×32 (CIFAR). Note that most of these models downsample the larger image size in the first few layers by using larger kernel sizes. As an example ResNet50 pretrained on ImageNet has 7×7 filters in the initial layers which will most likely give poor results for smaller image sizes. Be careful!!
- **(Optional)** Training your SSL algorithm on one subset of labeled data may not always give a correct comparison of your methods. Traditionally any SSL algorithm is run on 5 subsets of labeled data and the results are averaged. If you do this please mention the mean and standard deviation of the in your result tables.
- If your zip file size exceeds the limit of the allowed submission limit on MS-Teams then do the following. Create a zip file of your your project with a downloadable link to the zip file which is not password protected. You may use google drive, dropbox, etc for this. Compute the md5sum of this file and send us the downloadable link and the md5sum of the file along with your submission on MS-Teams before the deadline. We will generate the checksum for the zip file and match it with the checksum you provide us. You still need to submit your code on MS-teams without the large files and include the checksum and the link in the README that you create.

[1]https://github.com/szagoruyko/wide-residual-networks

Good luck!!

References

[1] Ekin D Cubuk et al. “Randaugment: Practical automated data augmentation with a reduced search space”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 702–703.

[2] Chengyue Gong, Dilin Wang, and Qiang Liu. “AlphaMatch: Improving Consistency for Semi-supervised Learning with Alpha-divergence”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 13683–13692.

[3] Zijian Hu et al. “SimPLE: Similar Pseudo Label Exploitation for Semi-Supervised Classification”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 15099–15108.

[4] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[5] Dong-Hyun Lee et al. “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks”. In: *Workshop on challenges in representation learning, ICML*. Vol. 3. 2. 2013, p. 896.

[6] Takeru Miyato et al. “Virtual adversarial training: a regularization method for supervised and semi-supervised learning”. In: *IEEE transactions on pattern analysis and machine intelligence* 41.8 (2018), pp. 1979–1993.

[7] Kihyuk Sohn et al. “Fixmatch: Simplifying semi-supervised learning with consistency and confidence”. In: *arXiv preprint arXiv:2001.07685* (2020).

[8] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *BMVC*. 2016.

[9] Bowen Zhang et al. “Flexmatch: Boosting semi-supervised learning with curriculum pseudo labeling”. In: *Advances in Neural Information Processing Systems* 34 (2021).

Submission instructions

The following instructions are mandatory. If you are not following them, tutors can decide to not correct your exercise.

- Please submit the assignment as a **team of two** students.
- Write the Microsoft Teams user name, student id and the name of each member of your team on your submission.
- Hand in zip file containing a report PDF with three directories made for each task. Do not include any data or cache files (e.g. __pycache__).
- Important: please name the submitted zip folder and files inside using the format: **Name1_id1_Name2_id2**.
- Your assignment solution must be uploaded by only **one** of your team members to the ’Assignments’ tab of the tutorial team (in **Microsoft Teams**). Please remember to press the **Hand In** button after uploading your work.
- If you have any trouble with the submission, contact your tutor **before** the deadline.