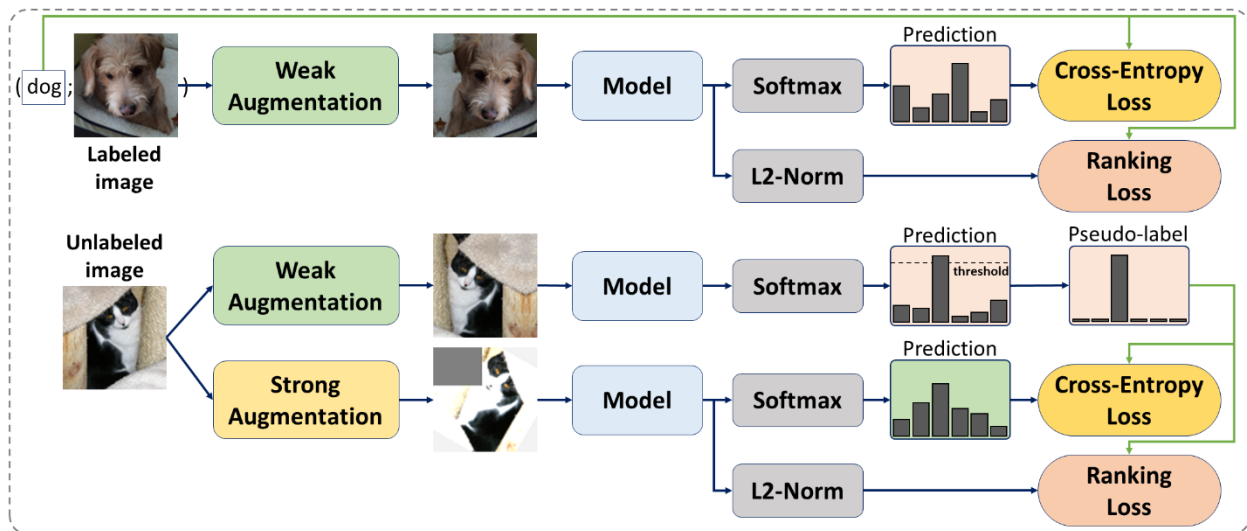


RankingMatch: Delving into Semi-Supervised Learning with Consistency Regularization and Ranking Loss

We propose RankingMatch, a novel semi-supervised learning (SSL) method which unifies the idea of semi-supervised learning and metric learning. Concretely, we utilize Ranking loss of metric learning to apply more constraints to the objective function of consistency regularization SSL method. Our method encourages the model to produce the similar outputs for not only the different perturbations of the same input but also the samples from the same class. The overall framework of RankingMatch is shown below.



Environment

The code was developed using Python 3.6. NVIDIA GPUs are needed. We recommend to use Anaconda. We already developed using Anaconda2. You can follow the steps below to set up the Anaconda environment.

1. Install Anaconda2: Go to <https://www.anaconda.com/distribution/#download-section>, copy `${LINK}` for appropriate version.

```
curl -O ${LINK}
bash ${DOWNLOADED_FILE}
source ~/.bashrc
```

2. Create new environment:

```
conda create --name RankingMatch python=3.6 -y
```

3. Activate the environment:

```
conda activate RankingMatch
```

Installation

1. Install pytorch >= v1.6.0 and torchvision following <https://pytorch.org/>.
2. Clone this repo, and the cloned directory is called as `${RANKINGMATCH_ROOT}`.
3. Install dependencies:

```
tensorboardx
cv2
scipy
matplotlib
scikit-learn
pandas
seaborn
```

Note: You can follow the steps below to install dependencies in the Anaconda2 environment:

```
conda install -c conda-forge tensorboardx
python3.6 -m pip install opencv-python==3.4.1.15
conda install scipy
conda install matplotlib
conda install -c anaconda scikit-learn
conda install -c anaconda pandas
conda install -c anaconda seaborn
```

Data preparation

For CIFAR-10 dataset, download and extract them under `${RANKINGMATCH_ROOT}/data`:

```
mkdir data          # if data folder has not existed
cd data
wget -c http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
tar -xzf cifar-10-python.tar.gz
```

For CIFAR-100 dataset, download and extract them under `${RANKINGMATCH_ROOT}/data`:

```
mkdir data          # if data folder has not existed
cd data
wget -c http://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
tar -xzf cifar-100-python.tar.gz
```

For SVHN and STL-10 dataset, the data will be automatically downloaded and placed under `${RANKINGMATCH_ROOT}/data` when training in the first time.

For Tiny ImageNet dataset, download data from GoogleDrive (https://drive.google.com/file/d/1WgU3qatv_Qn1xjuh9JnYKdsH5YeQembL/view) and extract them under `${RANKINGMATCH_ROOT}/data`. Then, run `python tiny_imagenet_restructure.py` to restructure the validation images to be able to use `torchvision.datasets.ImageFolder` class.

Pre-trained models

In our paper, the results are reported on 5 datasets (CIFAR-10, CIFAR-100, SVHN, STL-10, and Tiny ImageNet), including 5 tables. You can download all our trained models from GoogleDrive (https://drive.google.com/drive/folders/1qoQv8mcZ-l_Jp1fUHUDfXLOX58QH_Uo8?usp=sharing) and place them under `${RANKINGMATCH_ROOT}/trained_models`. Create `trained_models` directory if it has not existed: `mkdir trained_models`.

```
${RANKINGMATCH_ROOT}
|-- trained_models
    |-- Table1_CIFAR10
        |-- fixmatchRA_cifar10_40_128_1
        |-- fixmatchRA_cifar10_250_128_1
        |-- fixmatchRA_cifar10_4000_128_1
        |-- mixmatch_cifar10_40_128_1
        |-- mixmatch_cifar10_250_128_1
        |-- mixmatch_cifar10_4000_128_1
        |-- rankingmatchBM_cifar10_40_128_1
        |-- rankingmatchBM_cifar10_250_128_1
        |-- rankingmatchBM_cifar10_4000_128_1
        |-- rankingmatchCT_cifar10_40_128_1
        |-- rankingmatchCT_cifar10_250_128_1
        |-- rankingmatchCT_cifar10_4000_128_1
        |-- ...
    |-- Table1_CIFAR100
        |-- fixmatchRA_cifar100_400_128_1
        |-- fixmatchRA_cifar100_2500_128_1
        |-- fixmatchRA_cifar100_10000_128_1
        |-- mixmatch_cifar100_400_128_1
        |-- mixmatch_cifar100_2500_128_1
        |-- mixmatch_cifar100_10000_128_1
        |-- rankingmatchBM_cifar100_400_128_1
        |-- rankingmatchBM_cifar100_2500_128_1
        |-- rankingmatchBM_cifar100_10000_128_1
        |-- rankingmatchCT_cifar100_400_128_1
        |-- rankingmatchCT_cifar100_2500_128_1
        |-- rankingmatchCT_cifar100_10000_128_1
        |-- ...
    |-- Table2_CIFAR10
        |-- rankingmatchBM_cifar10_40_256_1
        |-- rankingmatchBM_cifar10_250_256_1
        |-- rankingmatchBM_cifar10_4000_256_1
        |-- rankingmatchCT_cifar10_40_256_1
        |-- rankingmatchCT_cifar10_250_256_1
        |-- rankingmatchCT_cifar10_4000_256_1
        |-- ...
    |-- Table2_CIFAR100
        |-- rankingmatchBM_cifar100_large_400_128_1
        |-- rankingmatchBM_cifar100_large_2500_128_1
        |-- rankingmatchBM_cifar100_large_10000_128_1
        |-- rankingmatchCT_cifar100_large_400_128_1
        |-- rankingmatchCT_cifar100_large_2500_128_1
        |-- rankingmatchCT_cifar100_large_10000_128_1
```

```

|-- ...
|-- Table3_SVHN
|-- fixmatchRA_svhn_40_128_1
|-- fixmatchRA_svhn_250_128_1
|-- fixmatchRA_svhn_1000_128_1
|-- mixmatch_svhn_40_128_1
|-- mixmatch_svhn_250_128_1
|-- mixmatch_svhn_1000_128_1
|-- rankingmatchBM_svhn_40_128_1
|-- rankingmatchBM_svhn_250_128_1
|-- rankingmatchBM_svhn_1000_128_1
|-- rankingmatchCT_svhn_40_128_1
|-- rankingmatchCT_svhn_250_128_1
|-- rankingmatchCT_svhn_1000_128_1
|-- ...
|-- Table4_STL10
|-- fixmatchRA_stl10_1000_128_0
|-- rankingmatchBM_stl10_1000_128_0
|-- rankingmatchCT_stl10_1000_128_0
|-- ...
|-- Table5_Ablation
|-- rankingmatchBA_cifar10_4000_128
|-- rankingmatchBH_cifar10_4000_128
|-- rankingmatchBM_cifar10_4000_128
|-- rankingmatchCT_cifar10_4000_128
|-- rankingmatchCT_cifar10_4000_128_no-L2Norm
|-- ...
`-- Tiny_ImageNet
|-- fixmatchRA_tinyimagenet_9000_128_1
|-- rankingmatchBM_tinyimagenet_9000_128_1
|-- rankingmatchCT_tinyimagenet_9000_128_1
|-- ...

```

Training and Validating

Note:

- For all commands, `--dataset` option should be `CIFAR10`, `CIFAR100`, `SVHN`, `STL10`, or `TinyImageNet` for CIFAR-10, CIFAR-100, SVHN, STL-10, or Tiny ImageNet dataset respectively.
- For validating commands, `--checkpoint` option is the trained model used for evaluation.
- For training commands, `--num_label` option is the number of labeled data used for training and `--epochs` is the number of training epochs.
- In default, Wide ResNet-28-2 network architecture with 1.5 million parameters will be used for CIFAR-10, CIFAR-100, and SVHN dataset; Wide ResNet-37-2 network with 5.9 million parameters will be used for STL-10 and Tiny ImageNet dataset.
- If you want to use Wide ResNet-28-2-Large network architecture with 26 million parameters, `--large_model` option should be raised in the command.

Validating on test set using trained models

1. For using default network architecture (Wide ResNet-28-2 or Wide ResNet-37-2):

```
python eval.py --dataset [DATASET] --checkpoint [CHECKPOINT]
```

2. For using Wide ResNet-28-2-Large network architecture:

```
python eval.py --large_model --dataset [DATASET] --checkpoint [CHECKPOINT]
```

Example:

```
python eval.py --dataset CIFAR10 --checkpoint  
trained_models/Table1_CIFAR10/rankingmatchBM_cifar10_4000_128_1
```

Training

1. For MixMatch:

```
python train_mixmatch.py --dataset [DATASET] --num_label [NUM_OF_LABELS] --  
epochs [NUM_OF_EPOCHS]
```

2. For FixMatchRA:

```
python train_fixmatch_ra.py --dataset [DATASET] --num_label [NUM_OF_LABELS] --  
epochs [NUM_OF_EPOCHS]
```

3. For our RankingMatch:

- RankingMatch with Triplet loss:

```
python train_rankingmatch_tl.py --dataset [DATASET] --num_label  
[NUM_OF_LABELS] --epochs [NUM_OF_EPOCHS] --triplet [TRIPLET_LOSS_FUNC]
```

Note: [TRIPLET_LOSS_FUNC] is `batchmean`, `batch_all`, or `batchhard` for BatchMean, BatchAll, or BatchHard Triplet loss respectively.

- RankingMatch with Contrastive loss:

```
python train_rankingmatch_ct.py --dataset [DATASET] --num_label  
[NUM_OF_LABELS] --epochs [NUM_OF_EPOCHS]
```

4. For using Wide ResNet-28-2-Large network architecture:

Simply add `--large_model` option as follows:

```
python train_rankingmatch_tl.py --large_model --dataset [DATASET] --num_label  
[NUM_OF_LABELS] --epochs [NUM_OF_EPOCHS] --triplet [TRIPLET_LOSS_FUNC]
```

Note: for STL-10 dataset, you can use `--fold` option for choosing the corresponding 1000-label split. For instance, the below command is for training on STL-10 dataset with the third 1000-label split.

```
python train_fixmatch_ra.py --dataset STL10 --num_label 1000 --fold 2 --epochs  
128
```

Example: the below command is for training RankingMatch with BatchMean Triplet loss.

```
python train_rankingmatch_t1.py --dataset CIFAR10 --num_label 250 --epochs 256 --triplet batchmean
```

Visualization

We provide the code for visualizing the "logits" scores and computing the confusion matrix of the models on the corresponding test sets. We use t-SNE visualization, which reduces the high-dimensional features to a reasonable dimension to help grasp the tendency of the learned models. Given a checkpoint, you can easily run visualization by using the following command:

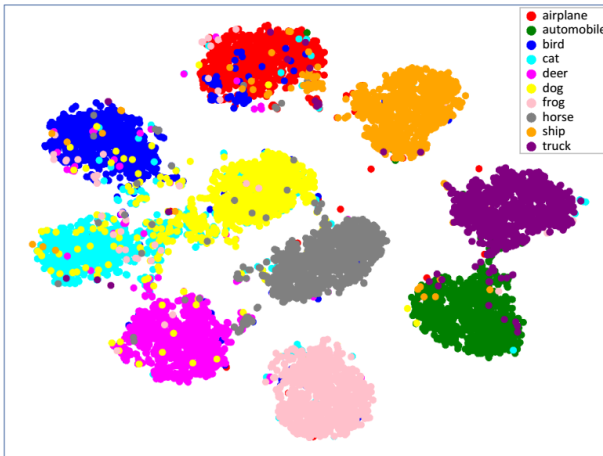
```
python vis.py --dataset [DATASET] --checkpoint [CHECKPOINT]
```

Example:

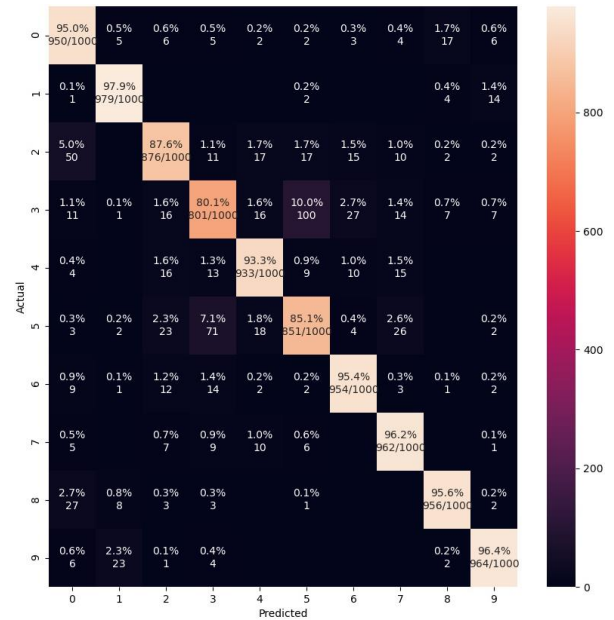
```
python vis.py --dataset CIFAR10 --checkpoint
trained_models/Table1_CIFAR10/fixmatchRA_cifar10_4000_128_3
```

The t-SNE visualization and confusion matrix will be saved in `${RANKINGMATCH_ROOT}/vis_dir` directory with the name corresponding to the checkpoint name. We show some visualizations of methods (trained on CIFAR-10 with 4000 labels and for 128 epochs) on CIFAR-10 test set as follows.

1. MixMatch: Test accuracy of 92.26%

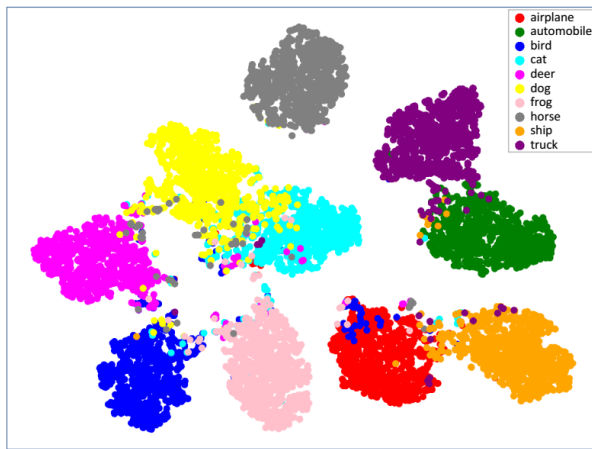


t-SNE visualization

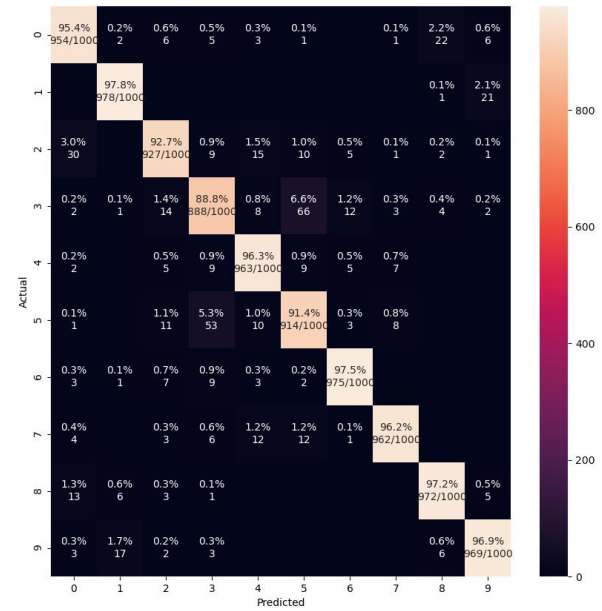


Confusion matrix

2. FixMatchRA: Test accuracy of 95.02%

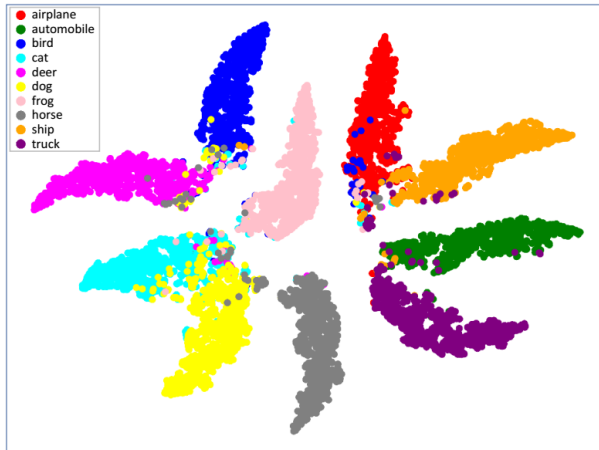


t-SNE visualization

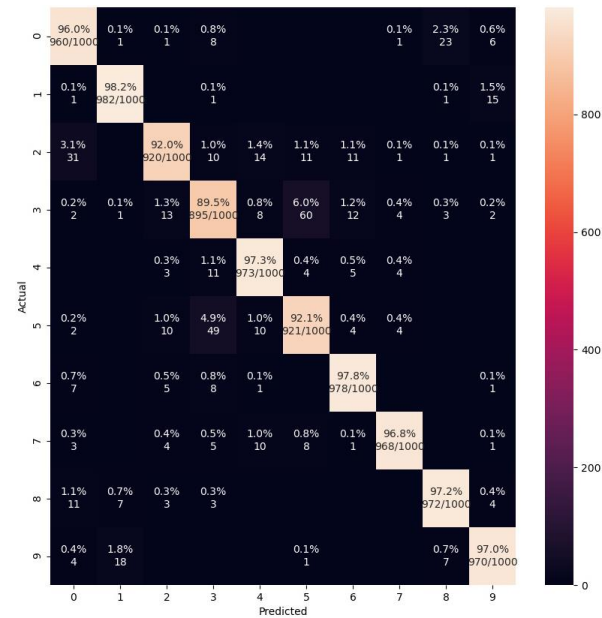


Confusion matrix

3. RankingMatchBM: Test accuracy of 95.39%

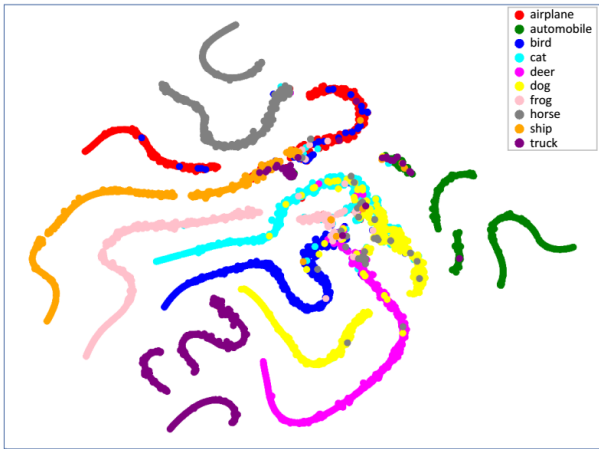


t-SNE visualization

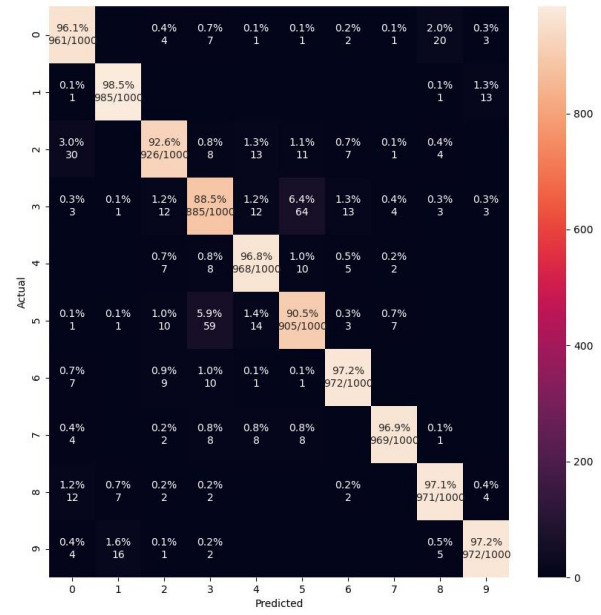


Confusion matrix

4. RankingMatchCT: Test accuracy of 95.14%



t-SNE visualization



Confusion matrix