PMP596 Homework 3

Due November 11th, Thursday, 6:00 PM

Problem 1: Two-stage: Faster/Mask R-CNN (40%)

1) Getting Started

In this homework assignment, we will use Detectron2 (Facebook) to help us to do the tasks of detection and segmentation.

Detectron2 is Facebook AI Research's next generation software system that implements state-of-the-art object detection algorithms. Here, we will go through some basic usage of detectron2, and finish the following tasks:

- Run inference on images, with existing pre-trained detectron2 models
- Train your own models on two custom datasets: traffic sign & balloon

Good luck and have fun!

2) Run a pretrained model

We first download some images from the given URLs. We can see there exists multiple objects in these images: bottles, microwaves, tables, refrigerators, people, etc. Let us see if we can detect them all by using a pre-trained model given by Detectron2.

downloaded images (input.jpg, test1.jpg, test2.jpg)

Let's take the first image (input.jpg) as an example. After applying the `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml`, a Faster R-CNN model which has a ResNet-50 backbone with the feature pyramid for dealing with different scales of objects and was pre-trained on COCO dataset with 80 different object categories, the model outputs are shown as the following, Let's take a look at the model output:

```
tensor([72, 39, 39, 39, 68, 45, 75, 55, 71, 41, 39, 75, 41, 45, 71, 41, 39],
       device='cuda:0')
Boxes(tensor([[  0.6764, 158.7422, 105.0817, 397.2280],
```

```
        [480.6438, 298.5876, 509.7858, 378.3471],
        [293.3246, 287.8804, 344.7385, 388.2297],
        [501.2999, 306.3395, 533.4362, 387.4037],
        [300.8816, 148.8771, 366.4546, 227.1448],
        [104.8806,  73.4484, 148.7521,  88.7021],
        [205.1707,  55.5022, 231.9424,  93.1075],
        [357.5101, 394.1846, 393.9808, 425.1449],
        [ 38.0390, 355.4640, 315.9145, 442.3582],
        [357.1301, 286.4151, 387.7237, 365.4046],
        [197.1212, 267.4070, 208.2965, 292.6432],
        [241.4115,  57.8996, 267.1059,  93.9473],
        [454.1900, 310.0651, 484.8028, 360.4231],
        [243.9083, 382.6956, 275.9801, 407.7662],
        [185.0177, 355.2176, 304.0785, 409.8493],
        [441.3940, 307.2240, 470.4242, 347.5514],
        [330.0779, 267.5388, 355.4027, 303.1052]], device='cuda:0'))
```

In **inference** mode, the model outputs a `list[dict]`, one dict for each image. For the object detection task, the dict contains the following fields:

* "instances": Instances object with the following fields:
  * "pred_boxes": Storing N boxes, one for each detected instance.
  * "scores": a vector of N scores.
  * "pred_classes": a vector of N labels in range [0, num_categories].

For more details, please see https://detectron2.readthedocs.io/tutorials/models.html#model output-format for specification. We can use "Visualizer" to draw the predictions on the image.

Visualization of model outputs

The model we just used is a two-stage Faster R-CNN detector, `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml`. Actually, the Detectron2 provides us more than that, you may find great amounts of models for different tasks in the given [MODEL_ZOO](https://github.com/facebookresearch/detectron2/tree/master/configs). Let's try to use some of these models and answer the following questions.

- Q1 (5%): Object Detection. Use the same configuration `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml`, with IoU threshold of 0.5 (`SCORE_THRESH_TEST=0.5`), to also run inference on the **rest two images** (test1.jpg & test2.jpg) and **view** the outputs with bounding boxes.
- Q2: Object Detection. Use the `COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml`, which has a ResNet-101 as the backbone, with IoU threshold of 0.5 and **view** the outputs of all three images with bounding boxes. By looking at the outputs, can you find the **difference** with the one `COCO-Detection/faster_rcnn_R_50_FPN_1x.yaml` we used in Q1? (e.g., numbers of objects, confidence scores, ...)
- Q3: Object Detection. Use the `COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml` with an IoU threshold of 0.9 and **view** the outputs of all three images with bounding boxes.
- Q4 (5%): Instance Segmentation. The models we have tried in Q1-Q3 are the Faster R-CNN models for *object detection*. Here, let's try a Mask R-CNN model `COCO-InstanceSegmentation/mask_rcnn_R_101_FPN_3x.yaml`, with IoU threshold of 0.5, to perform the *instance segmentation* and **view** the outputs of all three images with segmentation masks. **Compare** the difference of outputs between an object detection model with an instance segmentation model.

3) Train on a custom dataset - Faster R-CNN

You have already tried the pre-trained model on MS COCO datasets. Why not train your own model to perform customized detections? Here, let's **fine-tune** an existing Faster R-CNN model on a custom dataset in a new format.

We use the [traffic sign dataset] (https://www.dropbox.com/s/d8y6uc06027fpqo/traffic_sign_data.zip?dl=1). We'll train a traffic sign detection model from an existing model pre-trained on COCO dataset, available in detectron2's model zoo. Note that the COCO dataset does not have the "traffic sign" category, but we'll be able to recognize this new class in a few minutes.

a. Prepare the dataset.

Follow the instructions in `HW3_P1.ipynb` to download and unzip the data. You will find the file structures like this: traffic_sign_data/ ├── JPEGImages ├── labels ├── traffic_sign_test.txt └── traffic_sign_train.txt

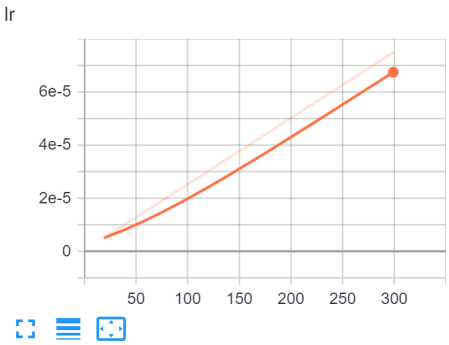
- `JPEGImages`: .jpg format images
- `labels`: corresponding .txt files that contain the bounding box label information
- `traffic_sign_train.txt`: a file that contains the path to the training images
- `traffic_sign_test.txt`: a file that contains the path to the testing images

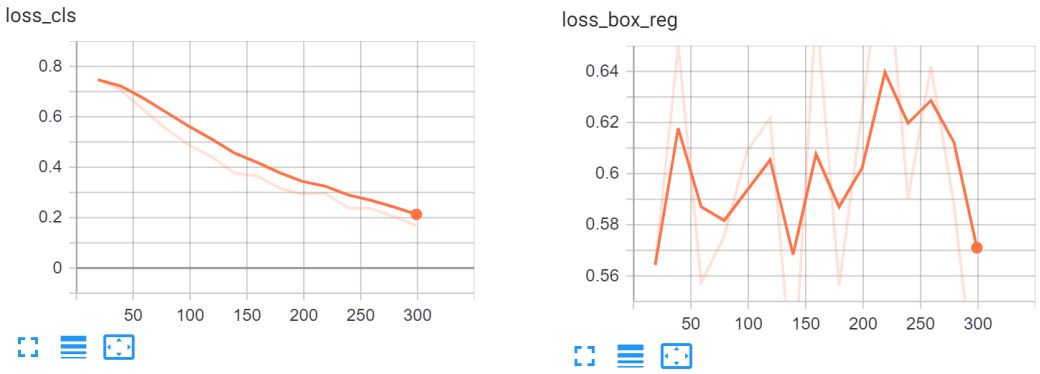
Here, the traffic sign dataset is in its custom dataset, therefore we write a function to parse it and prepare it into detectron2's standard format. See `` `get_traffic_sign_dicts` `` function in `HW3_P1.ipynb` for more details. To verify the data loading is correct, let's visualize the annotations of randomly selected samples in the training set:

Random visualization of *bbox* annotations of training samples

b. Train our own model.

Now, let's fine-tune a COCO-pretrained R50-FPN Faster-RCNN model on the traffic sign dataset. We use Tensorboard to visualize the training progress. The `total_loss` is the sum of 4 losses, RPN regression loss: `loss_rpn_loc`, RPN objectness loss: `loss_rpn_cls`, ROI regression loss: `loss_box_loc` and ROI classification loss: `loss_box_cls`. The x-axis represents the number of iterations, and y-axis represents the loss value. With the increase of iterations, you will notice the total_loss decreases.





Training Curves shown in the Tensorboard

c. Inference & Evaluation

Inference: Now let's run inference containing everything we've set previously. First, let's create a predictor using the model we just trained. Then, we randomly select several samples to visualize the prediction results.

Visualization of Faster R-CNN outputs

Evaluation: We can also evaluate its performance using the Average Precision (AP) metric.

```
Loading and preparing results...
DONE (t=0.01s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
COCOeval_opt.evaluate() finished in 0.08 seconds.
Accumulating evaluation results...
COCOeval_opt.accumulate() finished in 0.02 seconds.
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.353
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.570
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.409
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.120
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.309
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.496
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.253
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.517
```

```
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.573
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.311
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.585
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.686
[10/28 05:09:22 d2.evaluation.coco_evaluation]: Evaluation results for bbox:
|   AP   |  AP50  |  AP75  |  APs   |  APm   |  APl   |
|:------:|:------:|:------:|:------:|:------:|:------:|
| 35.349 | 57.048 | 40.929 | 12.027 | 30.909 | 49.640 |
OrderedDict([('bbox', {'AP': 35.34874923307184, 'AP50': 57.04849039231301, 'AP75':
40.92856309409448, 'APs': 12.026670579981444, 'APm': 30.909472073584805, 'APl':
49.639910640259735})])
```

The AP is ~35.3%. You may also see the detailed metrics for small, medium and large objects as well. Not bad! Here are something that I want you to try by yourself:

- Q5 (5%): **Change** the initial learning rate (`BASE_LR`) from `0.001` to `0.00025` and show the 4 training curves from the TensorBoard. By viewing the results (You may keep the rest of configurations fixed), does it improve the AP or not? Explain why.

- Q6 (5%): **Change** the number of iterations (`MAX_ITERS`) from `300` to `500` and show the 4 training curves from the Tensorboard. By viewing the results (You may keep the rest of configurations fixed), does it improve the AP or not? What about `1000`? Explain why.

- Q7 (5%): **Apply** the data augmentation techniques mentioned in HW2 to the training set and show the 4 training curves from the Tensorboard and view the AP performance. Does it improve the AP or not? Explain why.

4) Train on a custom dataset - Mask R-CNN

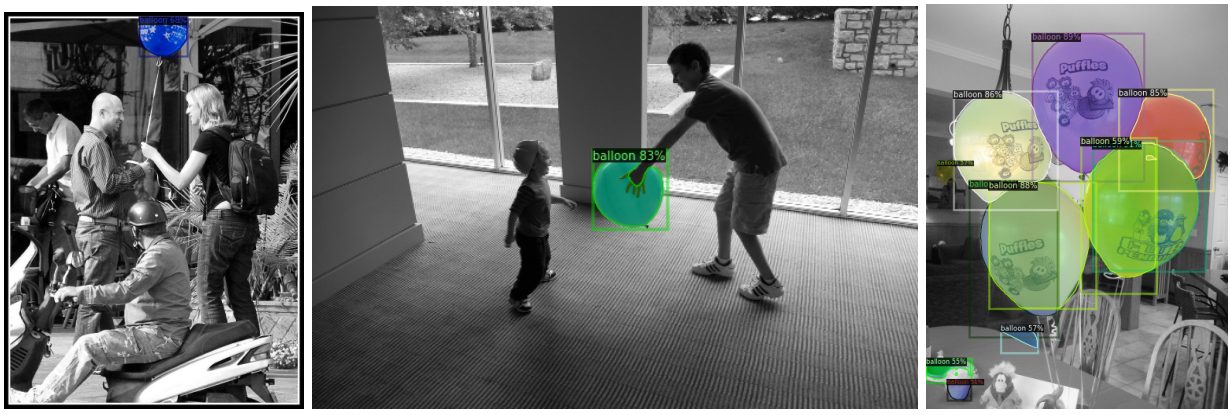
The above examples use Faster R-CNN to train on the traffic sign datasets to perform **object detection**. With few line modifications, we can train an **instance segmentation** model using Mask R-CNN as well. Notice that the traffic sign dataset only contains the bounding box labeling information, with no segmentation mask labeling, which is not enough to train a Mask R-CNN model. Due to this reason, we switch to another dataset: [balloon segmentation dataset](https://github.com/matterport/Mask_RCNN/tree/master/samples/balloon), which only has one class: balloon. Follow the instructions in `HW3_P1.ipynb` to download the balloon dataset and answer the following questions.

Write codes to load and visualize the balloon dataset in the similar manner. You need to take a careful look at the label files and construct your `` `get_balloon_dicts` `` functions to load extra poly mask information. If you load the dataset correctly, you will see training samples like the following.

Random visualization of *segm* annotations of training samples

- Q8 (10%): **Fine-tune** the pre-trained model `COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_1x.yaml` on the balloon dataset with the following configurations and **show** the TensorBoard Visualization.
    - IMS_BATCH_SIZE = 2
    - BASE_LR = 0.00025
    - MAX_ITER = 300
    - ROI_HEADS.BATCH_SIZE_PER_IMG = 128
    - ROI_HEADS.NUM_CLASSES = 1

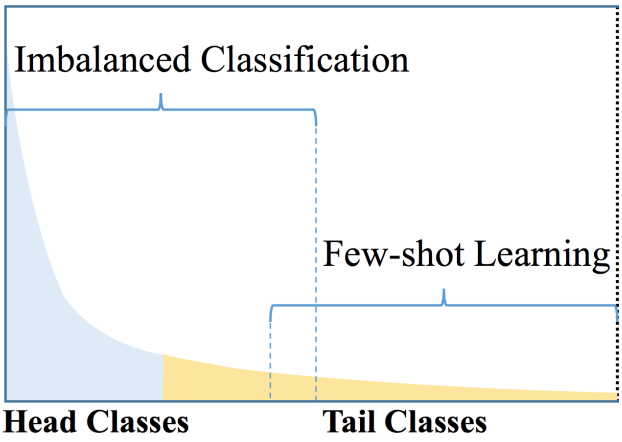
- Q9 (5%): Use your own trained model to do the **inference** on testing datasets, at least **plot 3** prediction results. Then, use the COCO API to **report** your testing Average Precision (AP). If your model is trained correctly, you will see the prediction results like the following figures.



Visualization of Mask R-CNN outputs

Problem 2: Image Classification on Imbalanced Dataset (20%)

In the existing visual recognition setting, the training data and testing data are both balanced under a closed-world setting, e.g., the ImageNet dataset. However, this setting is not a good proxy for the real-world scenario. For example, it is never possible for ecologists to gather balanced wildlife datasets because animal distribution is imbalanced. This imbalanced data distribution in the training set may largely degrade the performance of the machine learning or deep learning based method.



Our task of long-tailed recognition must learn from long-tail distributed training data and deal with imbalanced classification and few-shot learning over the entire spectrum.

Therefore, to faithfully reflect these aspects, we have to carefully study **Long-Tailed Recognition (LTR)** arising in natural data settings. A practical system shall be able to classify among a few common and many rare categories, to generalize the concept of a single category from only a few known instances. We define LTR as learning from **long-tail distributed training data** and evaluating the classification accuracy over **a balanced test set** which includes head and tail in a continuous spectrum (Fig. 1). Sometimes the training data can be so few, they cannot be used for training, and we have to resort to **few shot learning** methods.

Dataset The original version of CIFAR-10 and CIFAR-100 contains 50,000 training images and 10,000 testing images of size 32×32 with 10 and 100 classes, respectively. The original datasets and detailed descriptions can be found here: https://www.cs.toronto.edu/~kriz/cifar.html. For this question, we will use the CIFAR-50-LT dataset specifically collected from the CIFAR-100 dataset.

- Q1: Download dataset to your Google Drive from https://drive.google.com/drive/folders/1WGUKBP5Eta9DAltK1WtvRbX43iwP08DU?usp= sharing (please use your UW account to get access) and unzip the files. Please follow the instructions in the notebook to load/print the labels as well as visualize some images.

Method

- Q2 (5%): Train the CNN
  a. Use the CNN in HW2 to train the model on the balanced CIFAR50 dataset and train the CNN on the balanced CIFAR50 training set. **Evaluate and report** the classification accuracies on the testing set. Note: You can use any network configurations you implemented in HW2.
  b. Use the same CNN in HW2 to train the model on the **imbalanced** CIFAR50 dataset and Train the CNN on the imbalanced CIFAR50 training set. **Evaluate and report** the classification accuracies on the testing set.

- Q3 (15%): Implement tricks on your CNN for imbalanced dataset
  a. Before starting this question, please refer to the **Problem 4** (Paper Reading) for this homework: Bag of tricks for long-tailed visual recognition with deep convolutional neural networks. According to this paper, select **at least three** tricks to implement on the imbalanced CIFAR50 training, we have listed several possible tricks here for your reference, e.g.,
    (1) Re-sampling
    (2) Re-weighting
    (3) Mixed-up
    (4) Etc., ...
  b. **Evaluate and report** the classification performance on the CIFAR50 testing set.

Problem 3: Transfer Learning and Domain Adaptation (20%)

The goal of **domain adaptation** is to transfer the knowledge of a model to a different but related data distribution. The model is trained on a source dataset and applied to a target dataset (usually unlabeled). For Problem 3, the model will be trained on regular MNIST images, but we want to get good performance on MNIST with random color (without any labels).

Above: MNIST dataset (original), Bottom: MNIST-M dataset (with random colors, without labels)

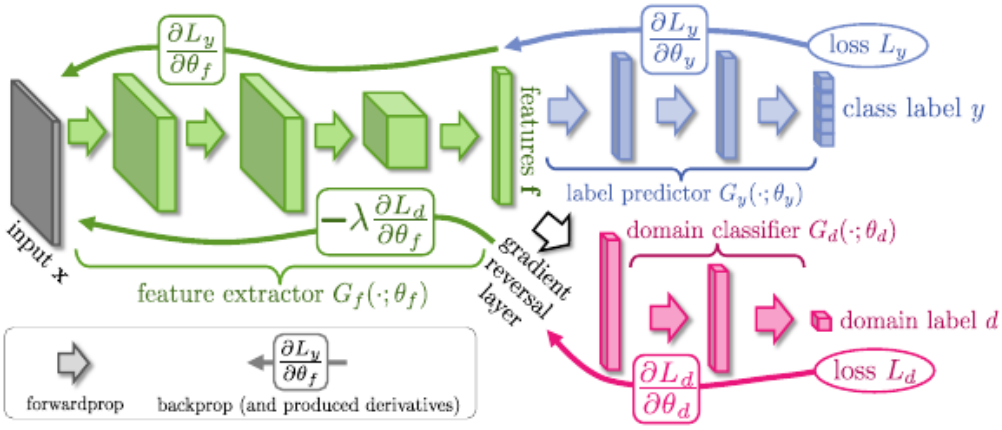
Problem Statement Given a labelled source domain (MNIST) and an unlabelled target domain (MNIST-M). We would like to train a classifier or a predictor which would give accurate predictions on the target domain.

Assumptions Probability distribution of source domain is not equal to the probability distribution of target domain. The conditional probability distribution of the labels given an instance from the source domain is equal to the conditional probability distribution of the labels given an instance from the target domain. Source dataset is labelled. Target dataset is unlabelled.

Approach Here, we adopt the DABP method mentioned in the paper "Unsupervised Domain Adaptation by Backpropagation".

- Feature Extractor (green): This is a neural network that will learn to perform the transformation on the source and target distribution.
- Label Classifier (blue): This is a neural network that will learn to perform the classification on the transformed source distribution. Since, the source domain is labelled.
- Domain Classifier (red): This is a neural network that will predict whether the output of the Feature Extractor is from the source distribution or the target distribution.

By using the above three components, the Feature Extractor will learn to produce discriminative and domain-invariant features.



Unsupervised Domain Adaptation by Backpropagation

(a) Follow the instructions in `HW3_P3.ipynb` to download the pre-processed MNIST and MNIST-M dataset and visualize some examples.

(b) The details of the implementation are also introduced in `HW3_P3.ipynb`. Please read it carefully before working on this problem. The implementation of the DABP includes the following components (please follow the provided sample codes):

(i) MNIST and MNITS-M DataLoader
(ii) Feature Extractor
(iii) Label and Domain Classifier
(iv) Gradient Reversal Layer

(c) Train the DABP model by running `` `main.py` `` and answer the following questions.

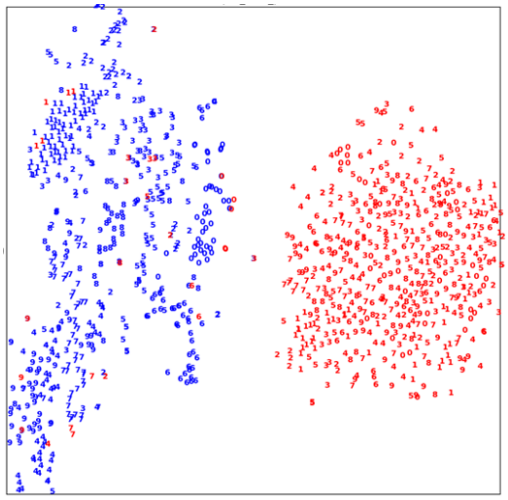
- Q1 (5%): **Perform 3** experiments on training and report your source and target accuracy in the tables below. (Your result is the average of the Target Accs. based on 3 experiments)

	Test1	Test2	Test3
Source Acc (%)	...		
Target Acc (%)	...		

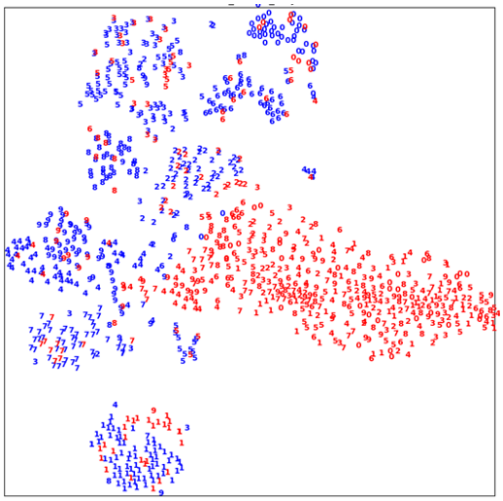
	Paper	Your Result

DABP (%)	52.25	...

- Q2 (5%): Write your own codes to **visualize** the feature space by using the `TSNE(perplexity=30, n_components=2, init='pca', n_iter=3000)`. Plot the feature distributions for both (1) original MNIST and MNIST-M inputs and (2) after DABP using source only. (You will find useful functions inside the `utils`function. The example plots are shown as the below.)



input_tsne_plots



after DABP using source only

- Q3 (10%): Discussions
  (1) From the results in Q2, are the both domains **closer/farther after performing the transformation**? If the answer is closer, it verifies that DABP can learn discriminative and domain invariant features. If not, explain your reasons.
  (2) List **one** of the main problems for the DABP method and **explain** why? (e.g., the gradient reversal layer, weights shared for both source domain and the target domain)

Problem 4: Paper Reading (20%)

Title: Bag of tricks for long-tailed visual recognition with deep convolutional neural networks
Authors: Yongshun Zhang, Xiu-Shen Wei, Boyan Zhou, Jianxin Wu
Abstract: In recent years, visual recognition on challenging long-tailed distributions, where classes often exhibit extremely imbalanced frequencies, has made great progress mostly based on various complex paradigms (e.g., meta learning). Apart from these complex methods, simple refinements on training procedures also make contributions. These refinements, also called tricks, are minor but effective, such as adjustments in the data distribution or loss functions. However, different tricks might conflict with each other. If users apply these long-tail related tricks inappropriately, it could cause worse recognition accuracy than expected. Unfortunately, there has not been a scientific guideline of these tricks in the literature. In this paper, we first collect existing tricks in long-tailed visual recognition and then perform extensive and systematic experiments, in order to give a detailed experimental guideline and obtain an effective combination of these tricks. Furthermore, we also propose a novel data augmentation approach based on class activation maps for long-tailed recognition, which can be friendly combined with re-sampling methods and shows excellent results. By assembling these tricks scientifically, we can outperform state-of-the-art methods on four long-tailed benchmark datasets, including ImageNet-LT and iNaturalist 2018.