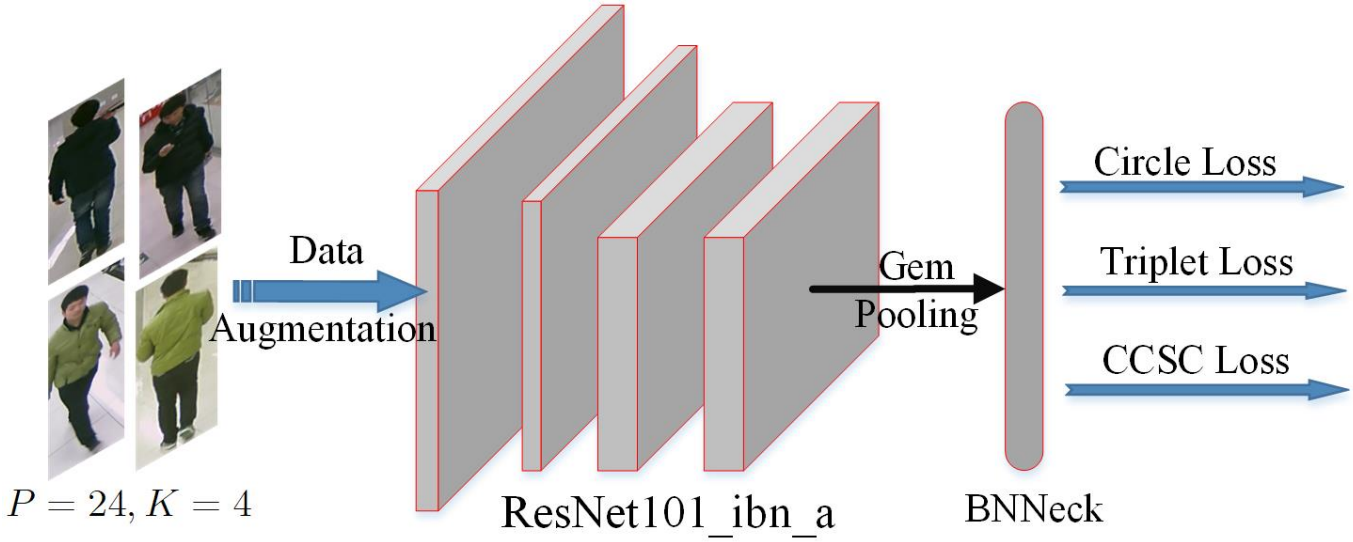
PRCV2020 Person Re-Identification Track:

The technical documentation of team **USTCZL**. If you have any question, please contact zwzhao98@mail.ustc.edu.cn for more details.

The trained model is saved in **PRCV/output/prcv20/baseline/final.pth**. The generated csv file is saved in **PRCV/output/prcv20/baseline/result.csv**.

Introduction

Our solution is mainly based on the Fast-ReID, VOC-ReID and StrongBaseline. The overall framework of our mothod is shown in the Figure below.



Characteristics

- [x] ResNet101_ibn_a + Circle Loss + GemPooling
- [x] FP16 training (30% faster with no precision drop)
- [x] Ranger optimizer to make training faster and better
- [x] Advanced data augmentations
- [x] Cross camera similarity constraint loss
- [x] Supporting global or local feature based networks: MGN and MSBA
- [x] Multi-model Ensembling (TODO)

Requirements

1. pytorch>=1.2.0
2. torchvision
3. yacs
4. python>=3.6
5. opencv-python (cv2)
6. apex (optional for FP16 training, if you don't have apex installed, please turn-off FP16 training by setting SOLVER.FP16=False. For testing or inference demo, the apex is not needed to be installed.)

```
$ git clone https://github.com/NVIDIA/apex
$ cd apex
$ pip install -v --no-cache-dir --global-option="--cpp_ext" --global-option="--
cuda_ext" ./
```

If you just want to call the demo, please jump to read the **Calling Interfaces** Section.

Train Stage

- Dataset Preparation

We prepare the PRCV2020 Re-ID dataset with following structure: `PRCV2020/PRID/train`, `PRCV2020/PRID/test` and you can check `data/dataset/prcv.py` for more details. please specify the root path of dataset to `ROOT_DIR` item in `configs/prcv_train.yml` and `configs/prcv_test.yml`.

- Training Script (`bash train.sh`)

```
python tools/train.py --config_file='configs/prcv_train.yml' MODEL.DEVICE_ID "
('0')"
```

- Generating Retrieval Results (CSV File) (`bash test.sh`)

Once the training process is finished, the trained model is saved in **output/prcv20/baseline/final.pth**. You can obtain the retrieval result (csv format) by running the following script. The generated csv file is saved in **output/prcv20/baseline/result.csv**. The `TEST.WEIGHT` denotes the path of trained model and you can specify it according to your requirements.

```
python tools/test.py --config_file='configs/prcv_test.yml' TEST.WEIGHT
'output/prcv20/baseline/final.pth' MODEL.DEVICE_ID "('0')"
```

Interface Implementation

We implemented 4 interface methods `init(), process(), get_Results(), release()` in **`tools/isee_interfaces.py`**. In the following section, we will give detailed comments about the 4 interface methods. We also give the demo about how to call these interfaces to inference. You can read **`tools/isee_interfaces.py`** for concrete implementations.

```


class PRCV2020REID(ISEEVisAlgIntf):

    def init(self, config_file, params_dict = None):
        """
        Load model.
        params:
          config_file: the path of the cofiguration file containing the
```

```
            necessary parameters (e.g., XML, json, YAML etc.).
            params_dict: the necessary parameters to initialize the project.
            It is in the type of dictionary as follows:
            {
              gpu_id: [-1], # the gpu id (a list of Integers), -1 means using CPU.
              model_path: ['/home/yourmodelpath', ..., ''], # a list of strings.
              reserved: {}  # other necessary parameters.
            }
          note:
            If overlapping prameters are existed in the configuration file and
            the variable of params_dict, the parameters in the variable of
            params_dict will be used.
          return:
            error code: 0 for success; a negative number for the ERROR type.
          """

          # read config file, yml format

          cfg.merge_from_file(config_file)
          cfg.merge_from_list(params_dict)
          cfg.freeze()

          # setting selected cuda devices
          if cfg.MODEL.DEVICE == "cuda":
              os.environ['CUDA_VISIBLE_DEVICES'] = cfg.MODEL.DEVICE_ID
          cudnn.benchmark = True

          # build model and loading trained model
          self.model = build_model(cfg, 1295)
          self.model.load_param(cfg.TEST.WEIGHT)

          # move the model to cuda and set model to eval mode
          self.model.to(cfg.MODEL.DEVICE)
          self.model.eval()

          # build image transform [ Resize, ToTensor, Normalize]
          self.transform = build_transforms(cfg, is_train=False)

          return 0
```

```
    def process(self, imgs_data, **kwargs):
          """
          Inference through loaded model.
          params:
            imgs_data: a list of images data (OpenCV  BGR format) [img1,img2,..imgN]
          return:
            error code: 0 for success; a negative number for the ERROR type.
          """

          feats = []
```

```
        with torch.no_grad():
            for i, img in enumerate(imgs_data):
                # convert cv2 format to PIL format
                img = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

                # transform PIL image to pytorch tensor [ Resize, ToTensor,
Normalize]
                img = self.transform(img).unsqueeze(0)

                # inference through the model
                img = img.cuda()
                feat = self.model(img)

                # whether to open test-time flip augmentation
                if cfg.TEST.FLIP_TEST:
                    img_flip = img.flip(dims=[3])  # NCHW
                    feat_flip = self.model(img_flip)
                    feat = (feat + feat_flip) / 2

                feats.append(feat)

        feats = torch.cat(feats, dim=0)
        # l2-normalizing feature vectors
        feats = torch.nn.functional.normalize(feats, dim=1, p=2) # N*2048
        feats = feats.cpu().numpy()

        # A list of output feature vectors (numpy format)
[feature1,feature2,...featureN]
        self.features  = [ feat for feat in feats]

        return 0
```

```
    def getResults(self, img_dir):
        """
        Get the processing results.
        params:
            img_dir: the directory which contains a list of images
        return:
            a list of l2-normlized feature vectors [feature1,feature2,...featureN]
        """
        # a list of image_names [img_name1,img_name2,..img_nameN]
        img_names = os.listdir(img_dir)
        # a list for storing opencv (BGR) format images data [img1,img2,..imgN]
        imgs_data = []

        # read the image by opencv
        for img_name in img_names:
            img = cv2.imread(os.path.join(img_dir,img_name)) # cv2 format: BGR
            imgs_data.append(img)

        # process the list of images by calling the function process()
```

```
        self.process(imgs_data)

        # return the l2-normalized feature vectors [feature1,feature2,...featureN]
        return self.features

    def release(self):
        print("release")
```

Calling Interfaces

Above, we give the detailed implementation codes and comments of the interfaces. Then, we provide the demo about how to use these interfaces. We claim that the trained model is placed in `output/prcv20/baseline/final.pth` and config_file is placed in `./configs/prcv_test.yml`

Demo script: (`bash demo.sh`)

```
python tools/isee_interface.py --config_file configs/prcv_test.yml --img_dir imgs
MODEL.DEVICE_ID "('0')"  TEST.WEIGHT "output/prcv20/baseline/final.pth"
```

The above script will read a list of images in `--img_dir` directory, load the trained model placed in `TEST.WEIGHT` path, perform the inference process and output a list of l2-normalized feature vectors.

You need to specify the path of config_file in `--config_file`, the path of trained model in `TEST.WEIGHT`, the path of images directory in `--img_dir`. Note that the `TEST.WEIGHT` is optional, if you don't specify it explicitly in the input command parameters, it will use the default path `output/prcv20/baseline/final.pth`.

For example, the `imgs` directory contains 2 images: RAP_REID_TEST_06700.png and RAP_REID_TEST_09063.png. By running the above demo script, you could obtain the corresponding extracted l2-normalized feature vectors: [feature1,feature2].

```
[array([-0.01007134, -0.00775381, -0.00682727, ..., -0.02445853,
       -0.00055178,  0.00954382], dtype=float32),
 array([-0.01623777, -0.02847788, -0.01604447, ..., -0.02845081,
       -0.01154274,  0.0082399 ], dtype=float32)]
```

Below we show the code to call these interfaces, you can modify them according to your requirements.

```
def main():
    parser = argparse.ArgumentParser(description="Re-ID model Inference Demo")
    parser.add_argument(
        "--config_file", default="./configs/prcv_test.yml", help="path to config
file", type=str
    )
    parser.add_argument(
        "--img_dir", default="imgs", help="path to the directory which contains a
list of images", type=str
```

```
    )
    parser.add_argument("opts", help="Modify config options using the command-
line", default=None,
                        nargs=argparse.REMAINDER)
    args = parser.parse_args()

    # Create an demo object which belongs the class PRCV2020REID (the children-
class of ISEEVisAlgIntf class)
    demo = PRCV2020REID()

    # Initialize the demo object by calling the function init (passing in the
config-file,args.opts)
    demo.init(config_file=args.config_file, params_dict=args.opts)

    # Calling the getResults() function to obtain the l2-normalized feature
vectors (passing in the img_dir)
    features = demo.getResults(args.img_dir) # [feature1,feature2,...,featureN]

    # print the features vectors [feature1,feature2,...,featureN] 2048-dim
    print(features)
```