torch$_{template}$

Release 0.0.4

Jan 14, 2020

Contents:

CHAPTER 1

DataLoader

1.1 Datasets

class torch_template.templates.dataloader.dataset.**ImageDataset**(*datadir*, *crop=None*, *aug=True*, *norm=False*)

ImageDataset for training.

Parameters

- **datadir** (*str*) – dataset root path, default input and label dirs are 'input' and 'gt'
- **crop** (*None, int or tuple*) – crop size
- **aug** (*bool*) – data argument (×8)
- **norm** (*bool*) – normalization

Example

train_dataset = ImageDataset('train', crop=256) for i, data in enumerate(train_dataset):

input, label, file_name = data

class torch_template.templates.dataloader.dataset.**ImageTestDataset**(*datadir*, *norm=False*)

ImageDataset for test.

Parameters

- **datadir** (*str*) – dataset path'
- **norm** (*bool*) – normalization

Example

test_dataset = ImageDataset('test', crop=256) for i, data in enumerate(test_dataset):

input, file_name = data

1.2 TTA

TTA plugin used in test data_loader loop, containing overlap and data_aug (8×)

Author: zks@tju.edu.cn

Refactor: xuhaoyu@tju.edu.cn

class torch_template.dataloader.tta.**OverlapTTA**(*img*, *nw*, *nh*, *patch_w=256*, *patch_h=256*, *norm_patch=False*, *flip_aug=False*, *device='cuda:0'*)

overlap TTA

Parameters

- **nw** (*int*) – num of patches (in width direction)
- **nh** (*int*) – num of patches (in height direction)
- **patch_w** (*int*) – width of a patch.
- **patch_h** (*int*) – height of a patch.
- **norm_patch** (*bool*) – if norm each patch or not.
- **flip_aug** (*bool*) – not used yet.
- **device** (*str*) – device string, default 'cuda:0'.

Usage Example

```
>>> from torch_template import OverlapTTA
>>> for i, data in enumerate(dataset):
>>>     tta = OverlapTTA(img, 10, 10, 256, 256, norm_patch=False, flip_
↪aug=False, device=opt.device)
>>>     for j, x in enumerate(tta):  # patch
>>>         generated = model(x)
>>>         torch.cuda.empty_cache()
>>>         tta.collect(generated[0], j)  # inference
>>>     output = tta.combine()
```

CHAPTER 2

Model_Zoo

```python
from .FFA import FFA
from .linknet import LinkNet
from .linknet import LinkNet50
from .unet import NestedUNet, UNet
from .pix2pix import GlobalGenerator, LocalEnhancer
from .transform_net import TransformNet
from .dense import Dense

model_zoo = {
    'FFA': FFA,
    'LinkNet': LinkNet,
    'LinkNet50': LinkNet50,
    'NestedUNet': NestedUNet,
    'UNet': UNet,
    'GlobalGenerator': GlobalGenerator,
    'LocalEnhancer': LocalEnhancer,
    'TransformNet': TransformNet,
    'Dense': Dense,
}

"""
  Model:       nf      n_params      256       512       256_batch8
   UNet         8       852,304       -        569M
NestedUNet     64     36,629,763    1851M     5365M        10037M
   FFA          -      4,455,913    5509M  out of memory out of memory
LinkNet50       -     28,762,115    1051M     1357M
 LinkNet        -     1,533,635      761M     1883M
  Global       64     45,614,595    1415M     1767M         2457M
TransformNet   32      1,673,097     829M     1587M         2615M
   Dense        ?     11,581,967     907M     1659M         2695M

"""
```

CHAPTER 3

Network

3.1 Model

```
import pdb

import numpy as np
import torch
import os

from torch import optim
import torch.nn.functional as F


from torch_template import model_zoo
from torch_template.loss.seg_loss import bce_loss, dice_loss, BCEFocalLoss

from torch_template.network.base_model import BaseModel
from torch_template.network.metrics import ssim, L1_loss
from torch_template.utils.torch_utils import ExponentialMovingAverage, print_network


models = {
    'Nested': model_zoo['NestedUNet']()

}


def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        m.weight.data.normal_(0.0, 0.02)
    elif classname.find('BatchNorm2d') != -1:
        m.weight.data.normal_(1.0, 0.02)
        m.bias.data.fill_(0)
```

(continues on next page)

(continued from previous page)

```
class Model(BaseModel):
    def __init__(self, opt):
        super(Model, self).__init__()
        self.opt = opt
        self.cleaner = models[opt.model].cuda(device=opt.device)
        ######################
        #    Init weights
        ######################
        self.cleaner.apply(weights_init)

        print_network(self.cleaner)

        self.g_optimizer = optim.Adam(self.cleaner.parameters(), lr=opt.lr)
        # self.d_optimizer = optim.Adam(cleaner.parameters(), lr=opt.lr)

        # load networks
        if opt.load:
            pretrained_path = opt.load
            self.load_network(self.cleaner, 'G', opt.which_epoch, pretrained_path)
            # if self.training:
            #     self.load_network(self.discriminitor, 'D', opt.which_epoch,␣
↪pretrained_path)

        self.avg_meters = ExponentialMovingAverage(0.95)
        self.save_dir = os.path.join(opt.checkpoint_dir, opt.tag)

    def update_G(self, img_var, y):
        opt = self.opt

        # cleaned = x
        cleaned = self.cleaner(img_var)

        ########################
        #       sigmoid
        ########################

        # cleaned = cleaned.mean(dim=1, keepdim=True)
        # y = y.mean(dim=1, keepdim=True)
        # f1 = f1_loss(cleaned, y, thresh=160/255)

        prediction = F.sigmoid(cleaned)
        target = F.sigmoid(y)

        ########################
        #       losses
        ########################
        bce = bce_loss(prediction, target) * opt.weight_bce

        dice = dice_loss(prediction, target) * opt.weight_dice
        l1 = L1_loss(prediction, target)

        # pdb.set_trace()
        loss = bce + dice

        # GAN loss
```

(continues on next page)

(continued from previous page)

```
        # loss_gen_adv = self.discriminitor.calc_gen_loss(input_fake=cleaned)
        self.avg_meters.update({'bce': bce.item(), 'dice': dice.item(), 'l1': l1.
↪item()})

        #loss_gen = loss + loss_gen_adv * 1.
        self.g_optimizer.zero_grad()
        loss.backward()
        self.g_optimizer.step()

        return cleaned

    def update_D(self, x, y):
        self.d_optimizer.zero_grad()
        # encode
        cleaned = self.cleaner(x)
        # h_b, n_b = self.gen_b.encode(x_b)
        # decode (cross domain)

        # D loss
        loss_dis = self.discriminitor.calc_dis_loss(input_fake=cleaned, input_real=y)
        self.avg_meters.update({'dis': loss_dis})

        loss_dis = loss_dis * 1.  # weights
        loss_dis.backward()
        self.d_optimizer.step()
        return cleaned

    def discriminate(self, input_label, test_image, use_pool=False):
        input_concat = torch.cat((input_label, test_image.detach()), dim=1)
        if use_pool:
            fake_query = self.fake_pool.query(input_concat)
            return self.netD.forward(fake_query)
        else:
            return self.netD.forward(input_concat)

    def forward(self, x):
        return self.cleaner(x)

    def inference(self, x, image=None):
        pass

    def save(self, which_epoch):
        self.save_network(self.cleaner, 'G', which_epoch)
        # self.save_network(self.discriminitor, 'D', which_epoch)

    def update_learning_rate(self):
        lrd = self.opt.lr / self.opt.niter_decay
        lr = self.old_lr - lrd
        for param_group in self.d_optimizer.param_groups:
            param_group['lr'] = lr
        for param_group in self.g_optimizer.param_groups:
            param_group['lr'] = lr
        if self.opt.verbose:
            print('update learning rate: %f -> %f' % (self.old_lr, lr))
        self.old_lr = lr
```

CHAPTER 4

Options

```
import argparse
import os

import torch

import misc_utils as utils

"""
    Arg parse
    opt = parse_args()
"""


def parse_args():
    # experiment specifics
    parser = argparse.ArgumentParser()

    parser.add_argument('--tag', type=str, default='cache',
                        help='folder name to save the outputs')
    parser.add_argument('--gpu_ids', type=str, default='0', help='gpu ids: e.g. 0  0,
↪1,2, 0,2. use -1 for CPU')
    parser.add_argument('--checkpoint_dir', type=str, default='./checkpoints', help=
↪'models are saved here')
    parser.add_argument('--log_dir', type=str, default='./logs', help='logs are saved␣
↪here')
    parser.add_argument('--result_dir', type=str, default='./results', help='results␣
↪are saved here')

    parser.add_argument('--model', type=str, default='Nested', help='which model to␣
↪use')
    parser.add_argument('--norm', type=str, default='instance',
                        help='[instance] normalization or [batch] normalization')

    # input/output sizes
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('--batch_size', type=int, default=1, help='input batch size')
    parser.add_argument('--resize', type=int, default=None, help='scale images to␣
↪this size')
    parser.add_argument('--crop', type=int, default=256, help='then crop to this size
↪')

    # for datasets
    parser.add_argument('--data_root', type=str, default='./datasets/')
    parser.add_argument('--dataset', type=str, default='train')
    parser.add_argument('--val_set', type=str, default=None)
    parser.add_argument('--test_set', type=str, default=None)

    # init weights
    parser.add_argument('--init', type=str, default=None, help='{normal, xavier,␣
↪kaiming, orthogonal}')

    # loss weight
    parser.add_argument('--weight_bce', type=float, default=20)
    parser.add_argument('--weight_dice', type=float, default=0.5)

    # training options
    parser.add_argument('--debug', action='store_true', help='debug mode')
    parser.add_argument('--load', type=str, default=None, help='load checkpoint')
    parser.add_argument('--which-epoch', type=int, default=0, help='which epoch to␣
↪resume')
    parser.add_argument('--epochs', type=int, default=500, help='epochs to train')
    parser.add_argument('--lr', type=float, default=0.0001, help='initial learning␣
↪rate for adam')

    parser.add_argument('--save_freq', type=int, default=50, help='freq to save models
↪')
    parser.add_argument('--eval_freq', '--val_freq', type=int, default=50, help='freq␣
↪to eval models')
    parser.add_argument('--log_freq', type=int, default=1, help='freq to vis in␣
↪tensorboard')

    return parser.parse_args()


opt = parse_args()

opt.device = 'cuda:' + opt.gpu_ids if torch.cuda.is_available() and opt.gpu_ids != '-1
↪' else 'cpu'

log_dir = os.path.join(opt.log_dir, opt.tag)
utils.try_make_dir(log_dir)
logger = utils.get_logger(f=os.path.join(log_dir, 'log.txt'), level='info')

logger.info('==================Options==================')
for k, v in opt._get_kwargs():
    logger.info(str(k) + '=' + str(v))
logger.info('===========================================')
# utils.print_args(opt)
```

CHAPTER 5

Utils

5.1 Torch_Utils

Misc PyTorch utils

Usage:

```
>>> from torch_template import torch_utils
>>> torch_utils.func_name()  # to call functions in this file
```

class torch_template.utils.torch_utils.**AverageMeters**(*dic=None*, *total_num=None*)

AverageMeter class

Example

```
>>> avg_meters = AverageMeters()
>>> for i in range(100):
>>>     avg_meters.update({'f': i})
>>>     print(str(avg_meters))
```

class torch_template.utils.torch_utils.**ExponentialMovingAverage**(*decay=0.9*, *dic=None*, *total_num=None*)

EMA class

Example

```
>>> ema_meters = ExponentialMovingAverage(0.98)
>>> for i in range(100):
>>>     ema_meters.update({'f': i})
>>>     print(str(ema_meters))
```

class torch_template.utils.torch_utils.**LR_Scheduler**(*mode*, *base_lr*, *num_epochs*, *iters_per_epoch=0*, *lr_step=0*, *warmup_epochs=0*, *logger=None*)

Learning Rate Scheduler

Example

```
>>> scheduler = LR_Scheduler('cos', opt.lr, opt.epochs, len(dataloader), warmup_
↪epochs=20)
>>> for i, data in enumerate(dataloader)
>>>     scheduler(self.g_optimizer, i, epoch)
```

Step mode: `lr = baselr * 0.1 ^ {floor(epoch-1 / lr_step)}` lr_step, lr0.1

Cosine mode: `lr = baselr * 0.5 * (1 + cos(iter/maxiter))`

Poly mode: `lr = baselr * (1 - iter/maxiter) ^ 0.9`

iters_per_epoch: number of iterations per epoch

torch_template.utils.torch_utils.**clamp**(*x*, *min=0.01*, *max=0.99*)
clamp a tensor.

Parameters

- **x** (*torch.Tensor*) – input tensor
- **min** (*float*) – value < min will be set to min.
- **max** (*float*) – value > max will be set to max.

Returns a clamped tensor.

Return type (torch.Tensor)

torch_template.utils.torch_utils.**create_summary_writer**(*log_dir*)
Create a tensorboard summary writer.

Parameters **log_dir** – log directory.

Returns a summary writer.

Return type (SummaryWriter)

Example

```
>>> writer = create_summary_writer(os.path.join(self.basedir, 'logs'))
>>> write_meters_loss(writer, 'train', avg_meters, iteration)
>>> write_loss(writer, 'train', 'F1', 0.78, iteration)
>>> write_image(writer, 'train', 'input', img, iteration)
>>> # shell
>>> tensorboard --logdir {base_path}/logs
```

torch_template.utils.torch_utils.**load_ckpt**(*model*, *ckpt_path*)
Load checkpoint.

Parameters

- **model** (*nn.Module*) – object of a subclass of nn.Module.
- **ckpt_path** (*str*) – *.pt file to load.

Example

```
>>> class Model(nn.Module):
>>>     pass
>>>
>>> model = Model().cuda()
>>> load_ckpt(model, 'model.pt')
```

torch_template.utils.torch_utils.**print_network**(*net: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa390>*, *print_size=False*)

Print network structure and number of parameters.

Parameters

- **net** (*nn.Module*) – network model.
- **print_size** (*bool*) – print parameter num of each layer.

Example

```
>>> import torchvision as tv
>>> from torch_template import torch_utils
>>>
>>> vgg16 = tv.models.vgg16()
>>> torch_utils.print_network(vgg16)
>>> '''
>>> features.0.weight [3, 64, 3, 3]
>>> features.2.weight [64, 64, 3, 3]
>>> features.5.weight [64, 128, 3, 3]
>>> features.7.weight [128, 128, 3, 3]
>>> features.10.weight [128, 256, 3, 3]
>>> features.12.weight [256, 256, 3, 3]
>>> features.14.weight [256, 256, 3, 3]
>>> features.17.weight [256, 512, 3, 3]
>>> features.19.weight [512, 512, 3, 3]
>>> features.21.weight [512, 512, 3, 3]
>>> features.24.weight [512, 512, 3, 3]
>>> features.26.weight [512, 512, 3, 3]
>>> features.28.weight [512, 512, 3, 3]
>>> classifier.0.weight [25088, 4096]
>>> classifier.3.weight [4096, 4096]
>>> classifier.6.weight [4096, 1000]
>>> Total number of parameters: 138,357,544
>>> '''
```

torch_template.utils.torch_utils.**repeat**(*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa3c8>*, **sizes*)

Repeat a dimension of a tensor.

Parameters

- **x** (*torch.Tensor*) – input tensor.
- **sizes** – repeat times for each dimension.

Returns a repeated tensor.

Return type (torch.Tensor)

Example

```
>>> t = repeat(t, 1, 3, 1, 1)  # same as t = t.repeat(1, 3, 1, 1) or t =␣
↪torch.cat([t, t, t], dim=1)
```

torch_template.utils.torch_utils.**save_ckpt**(*model*, *ckpt_path*)

Save checkpoint.

Parameters

- **model** (*nn.Module*) – object of a subclass of nn.Module.
- **ckpt_path** (*str*) – *.pt file to save.

Example

```
>>> class Model(nn.Module):
>>>     pass
>>>
>>> model = Model().cuda()
>>> save_ckpt(model, 'model.pt')
```

torch_template.utils.torch_utils.**tensor2im**(*x: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa2e8>*, *norm=False*, *to_save=False*)

Convert tensor to image.

Parameters

- **x** (*torch.Tensor*) – input tensor, [n, c, h, w] float32 type.
- **norm** (*bool*) – if the tensor should be denormed first
- **to_save** (*bool*) – if False, a float32 image of [h, w, c], if True, a uint8 image of [h, w, c].

Returns an image in shape of [h, w, c] if to_save else [c, h, w].

torch_template.utils.torch_utils.**write_graph**(*writer: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa1d0>*, *model*, *inputs_to_model=None*)

Write net graph into writer.

Parameters

- **writer** (*SummaryWriter*) – writer created by create_summary_writer()
- **model** (*nn.Module*) – model.
- **inputs_to_model** (*tuple or list*) – forward inputs.

Example

```
>>> from tensorboardX import SummaryWriter
>>> input_data = Variable(torch.rand(16, 3, 224, 224))
>>> vgg16 = torchvision.models.vgg16()
>>>
>>> writer = SummaryWriter(log_dir='logs')
>>> write_graph(vgg16, (input_data,))
```

torch_template.utils.torch_utils.**write_image**(*writer: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa1d0>*, *prefix*, *image_name: str*, *img*, *iteration*, *dataformats='CHW'*)

Write images into writer.

Parameters

- **writer** (*SummaryWriter*) – writer created by create_summary_writer()
- **prefix** (*str*) – any string, e.g. 'train'.
- **image_name** (*str*) – image name.
- **img** – image tensor in [C, H, W] shape.
- **iteration** (*int*) – epochs or iterations.
- **dataformats** (*str*) – 'CHW' or 'HWC' or 'NCHW'.

Example

```
>>> write_image(writer, 'train', 'input', img, iteration)
```

torch_template.utils.torch_utils.**write_loss**(*writer: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa1d0>*, *prefix*, *loss_name: str*, *value: float*, *iteration*)

Write loss into writer.

Parameters

- **writer** (*SummaryWriter*) – writer created by create_summary_writer()
- **prefix** (*str*) – any string, e.g. 'train'.
- **loss_name** (*str*) – loss name.
- **value** (*float*) – loss value.
- **iteration** (*int*) – epochs or iterations.

Example

```
>>> write_loss(writer, 'train', 'F1', 0.78, iteration)
```

torch_template.utils.torch_utils.**write_meters_loss**(*writer: <sphinx.ext.autodoc.importer._MockObject object at 0x7f24e70aa1d0>*, *prefix*, *avg_meters: torch_template.utils.torch_utils.Meters*, *iteration*)

Write all losses in a meter class into writer.

Parameters

- **writer** (*SummaryWriter*) – writer created by create_summary_writer()
- **prefix** (*str*) – any string, e.g. 'train'.
- **avg_meters** (AverageMeters *or* ExponentialMovingAverage) – meters.
- **iteration** (*int*) – epochs or iterations.

Example

```
>>> writer = create_summary_writer(os.path.join(self.basedir, 'logs'))
>>> ema_meters = ExponentialMovingAverage(0.98)
>>> for i in range(100):
>>>     ema_meters.update({'f1': i, 'f2': i*0.5})
>>>     write_meters_loss(writer, 'train', ema_meters, i)
```

5.2 Misc_Utils

Misc_utils doc can be found here.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index