"""General-purpose training script for image-to-image translation.

This script works for various models (with option '--model': e.g., pix2pix, cyclegan, colorization) and

different datasets (with option '--dataset\_mode': e.g., aligned, unaligned, single, colorization).

You need to specify the dataset ('--dataroot'), experiment name ('--name'), and model ('--model').

It first creates model, dataset, and visualizer given the option.

It then does standard network training. During the training, it also visualize/save the images, print/save the loss plot, and save models.

The script supports continue/resume training. Use '--continue\_train' to resume your previous training.

Example:

Train a CycleGAN model:

python train.py --dataroot ./datasets/maps --name maps\_cyclegan --model cycle\_gan

Train a pix2pix model:

python train.py --dataroot ./datasets/facades --name facades\_pix2pix --model pix2pix --direction BtoA

See options/base\_options.py and options/train\_options.py for more training options.

See training and test tips at: https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/docs/tips.md

See frequently asked questions at: https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/blob/master/docs/qa.md

"""

import time

from options.train\_options import TrainOptions

from data import create\_dataset

from models import create\_model

from util.visualizer import Visualizer

if \_\_name\_\_ == '\_\_main\_\_':

opt = TrainOptions().parse() # get training options

dataset = create\_dataset(opt) # create a dataset given opt.dataset\_mode and other options

dataset\_size = len(dataset) # get the number of images in the dataset.

print('The number of training images = %d' % dataset\_size)

model = create\_model(opt) # create a model given opt.model and other options

model.setup(opt) # regular setup: load and print networks; create schedulers

visualizer = Visualizer(opt) # create a visualizer that display/save images and plots

total\_iters = 0 # the total number of training iterations

for epoch in range(opt.epoch\_count, opt.n\_epochs + opt.n\_epochs\_decay + 1): # outer loop for different epochs; we save the model by <epoch\_count>, <epoch\_count>+<save\_latest\_freq>

epoch\_start\_time = time.time() # timer for entire epoch

iter\_data\_time = time.time() # timer for data loading per iteration

epoch\_iter = 0 # the number of training iterations in current epoch, reset to 0 every epoch

visualizer.reset() # reset the visualizer: make sure it saves the results to HTML at least once every epoch

model.update\_learning\_rate() # update learning rates in the beginning of every epoch.

for i, data in enumerate(dataset): # inner loop within one epoch

iter\_start\_time = time.time() # timer for computation per iteration

if total\_iters % opt.print\_freq == 0:

t\_data = iter\_start\_time - iter\_data\_time

total\_iters += opt.batch\_size

epoch\_iter += opt.batch\_size

model.set\_input(data) # unpack data from dataset and apply preprocessing

model.optimize\_parameters() # calculate loss functions, get gradients, update network weights

if total\_iters % opt.display\_freq == 0: # display images on visdom and save images to a HTML file

save\_result = total\_iters % opt.update\_html\_freq == 0

model.compute\_visuals()

visualizer.display\_current\_results(model.get\_current\_visuals(), epoch, save\_result)

if total\_iters % opt.print\_freq == 0: # print training losses and save logging information to the disk

losses = model.get\_current\_losses()

t\_comp = (time.time() - iter\_start\_time) / opt.batch\_size

visualizer.print\_current\_losses(epoch, epoch\_iter, losses, t\_comp, t\_data)

if opt.display\_id > 0:

visualizer.plot\_current\_losses(epoch, float(epoch\_iter) / dataset\_size, losses)

if total\_iters % opt.save\_latest\_freq == 0: # cache our latest model every <save\_latest\_freq> iterations

print('saving the latest model (epoch %d, total\_iters %d)' % (epoch, total\_iters))

save\_suffix = 'iter\_%d' % total\_iters if opt.save\_by\_iter else 'latest'

model.save\_networks(save\_suffix)

iter\_data\_time = time.time()

if epoch % opt.save\_epoch\_freq == 0: # cache our model every <save\_epoch\_freq> epochs

print('saving the model at the end of epoch %d, iters %d' % (epoch, total\_iters))

model.save\_networks('latest')

model.save\_networks(epoch)

print('End of epoch %d / %d \t Time Taken: %d sec' % (epoch, opt.n\_epochs + opt.n\_epochs\_decay, time.time() - epoch\_start\_time))