import logging

import argparse

import time

import os

import pickle

import numpy as np

from logger import Logger

import sys

from collections import OrderedDict

from tensorboardX import SummaryWriter

import torch

import torch.nn as nn

from torch.utils.data import Dataset, DataLoader

from torch.optim.lr\_scheduler import ReduceLROnPlateau

from network.rtpose\_vgg import get\_model, use\_vgg

#####################################

# DATASET #

#####################################

class Pose\_Dataset(Dataset):

def \_\_init\_\_(self, data\_dir, num\_im):

super(Pose\_Dataset, self).\_\_init\_\_()

data\_files = os.listdir(data\_dir)

self.data\_paths = [os.path.join(data\_dir, d) for d in data\_files]

self.data\_paths = self.data\_paths[0:num\_im]

def \_\_len\_\_(self):

return len(self.data\_paths)

def \_\_getitem\_\_(self, index):

with open(self.data\_paths[index], 'rb') as f:

sample = pickle.load(f)

return sample

logger = Logger('./logs')

#sys.argv=[''];

#del sys

# Hyper-params

parser = argparse.ArgumentParser(description='PyTorch rtpose Training')

s = '/scratch/aek495/pyenv2/pytorch\_Pose/'

parser.add\_argument('--data\_dir', default=s+'training/dataset/COCO/images/', type=str, metavar='DIR',

help='path to where coco images stored')

parser.add\_argument('--preproc\_dir', default=s+'training/dataset/COCO/preprocess', type=str, metavar='DIR',

help='path to where coco images preprocessed')

parser.add\_argument('--valid\_dir', default=s+'training/dataset/COCO/preprocess/valid', type=str, metavar='DIR',

help='path to preprocessed valid images')

parser.add\_argument('--train\_dir', default=s+'training/dataset/COCO/preprocess/train', type=str, metavar='DIR',

help='path to preprocessed train images')

parser.add\_argument('--mask\_dir', default=s+'training/dataset/COCO/mask/', type=str, metavar='DIR',

help='path to where coco images stored')

parser.add\_argument('--logdir', default=s+'logs/', type=str, metavar='DIR',

help='path to where tensorboard log restore')

parser.add\_argument('--json\_path', default=s+'training/dataset/COCO/COCO.json', type=str, metavar='PATH',

help='path to where coco images stored')

parser.add\_argument('--model\_path', default=s+'network/weight/', type=str, metavar='DIR',

help='path to where the model saved')

parser.add\_argument('--lr', '--learning-rate', default=1., type=float,

metavar='LR', help='initial learning rate')

parser.add\_argument('--momentum', default=0.9, type=float, metavar='M',

help='momentum')

parser.add\_argument('--num\_train', default=50, type=int, metavar='NT',

help='number of training images')

parser.add\_argument('--num\_valid', default=10, type=int, metavar='NV',

help='number of validation images')

parser.add\_argument('--epochs', default=200, type=int, metavar='N',

help='number of total epochs to run')

parser.add\_argument('--weight-decay', '--wd', default=0.000, type=float,

metavar='W', help='weight decay (default: 1e-4)')

parser.add\_argument('--nesterov', dest='nesterov', action='store\_true')

parser.add\_argument('-o', '--optim', default='sgd', type=str)

#Device options

parser.add\_argument('--gpu\_ids', dest='gpu\_ids', help='which gpu to use', nargs="+",

default=[0,1,2,3], type=int)

parser.add\_argument('--batch\_size', default=80, type=int,

metavar='N', help='mini-batch size (default: 256)')

parser.add\_argument('--print\_freq', default=20, type=int, metavar='N',

help='number of iterations to print the training statistics')

args = parser.parse\_args()

os.environ['CUDA\_VISIBLE\_DEVICES'] = ','.join(str(e) for e in args.gpu\_ids)

params\_transform = dict()

params\_transform['mode'] = 5

# === aug\_scale ===

params\_transform['scale\_min'] = 0.5

params\_transform['scale\_max'] = 1.1

params\_transform['scale\_prob'] = 1

params\_transform['target\_dist'] = 0.6

# === aug\_rotate ===

params\_transform['max\_rotate\_degree'] = 40

# ===

params\_transform['center\_perterb\_max'] = 40

# === aug\_flip ===

params\_transform['flip\_prob'] = 0.5

params\_transform['np'] = 56

params\_transform['sigma'] = 7.0

params\_transform['limb\_width'] = 1.

def build\_names():

names = []

for j in range(1, 7):

for k in range(1, 3):

names.append('loss\_stage%d\_L%d' % (j, k))

return names

def get\_loss(saved\_for\_loss, heat\_temp, heat\_weight,

vec\_temp, vec\_weight):

names = build\_names()

saved\_for\_log = OrderedDict()

criterion = nn.MSELoss(size\_average=True).cuda()

#criterion = encoding.nn.DataParallelCriterion(criterion, device\_ids=args.gpu\_ids)

total\_loss = 0

for j in range(6):

pred1 = saved\_for\_loss[2 \* j] \* vec\_weight

"""

print("pred1 sizes")

print(saved\_for\_loss[2\*j].data.size())

print(vec\_weight.data.size())

print(vec\_temp.data.size())

"""

gt1 = vec\_temp \* vec\_weight

pred2 = saved\_for\_loss[2 \* j + 1] \* heat\_weight

gt2 = heat\_weight \* heat\_temp

"""

print("pred2 sizes")

print(saved\_for\_loss[2\*j+1].data.size())

print(heat\_weight.data.size())

print(heat\_temp.data.size())

"""

# Compute losses

loss1 = criterion(pred1, gt1) \* 0

# if j == 0:

loss2 = criterion(pred2, gt2)

#else:

# loss2 = criterion(pred2, gt2) \* 0

total\_loss += loss1

total\_loss += loss2

# print(total\_loss)

# Get value from Variable and save for log

saved\_for\_log[names[2 \* j]] = loss1.item()

saved\_for\_log[names[2 \* j + 1]] = loss2.item()

saved\_for\_log['max\_ht'] = torch.max(

saved\_for\_loss[-1].data[:, 0:-1, :, :]).item()

saved\_for\_log['min\_ht'] = torch.min(

saved\_for\_loss[-1].data[:, 0:-1, :, :]).item()

saved\_for\_log['max\_paf'] = torch.max(saved\_for\_loss[-2].data).item()

saved\_for\_log['min\_paf'] = torch.min(saved\_for\_loss[-2].data).item()

return total\_loss, saved\_for\_log

def train(train\_loader, model, optimizer, epoch):

batch\_time = AverageMeter()

data\_time = AverageMeter()

losses = AverageMeter()

meter\_dict = {}

for name in build\_names():

meter\_dict[name] = AverageMeter()

meter\_dict['max\_ht'] = AverageMeter()

meter\_dict['min\_ht'] = AverageMeter()

meter\_dict['max\_paf'] = AverageMeter()

meter\_dict['min\_paf'] = AverageMeter()

# switch to train mode

model.train()

end = time.time()

print("training")

nb\_found = 0

for i, (img, heatmap\_target, heat\_mask, paf\_target, paf\_mask) in enumerate(train\_loader):

#print(img.size())

if type(img[0]) == 'str':

# print('in here')

continue

else:

nb\_found += 1

# print(nb\_found, i)

# measure data loading time

#writer.add\_text('Text', 'text logged at step:' + str(i), i)

#for name, param in model.named\_parameters():

# writer.add\_histogram(name, param.clone().cpu().data.numpy(),i)

data\_time.update(time.time() - end)

img = img.squeeze(1).cuda()

heatmap\_target = heatmap\_target.squeeze(1).cuda()

heat\_mask = heat\_mask.squeeze(1).cuda()

paf\_target = paf\_target.squeeze(1).cuda()

paf\_mask = paf\_mask.squeeze(1).cuda()

#img = img.squeeze(1)

#heatmap\_target = heatmap\_target.squeeze(1)

#heat\_mask = heat\_mask.squeeze(1)

#paf\_target = paf\_target.squeeze(1)

#paf\_mask = paf\_mask.squeeze(1)

# compute output

\_,saved\_for\_loss = model(img)

total\_loss, saved\_for\_log = get\_loss(saved\_for\_loss, heatmap\_target, heat\_mask,

paf\_target, paf\_mask)

for name,\_ in meter\_dict.items():

meter\_dict[name].update(saved\_for\_log[name], img.size(0))

losses.update(total\_loss, img.size(0))

# compute gradient and do SGD step

optimizer.zero\_grad()

total\_loss.backward()

optimizer.step()

# measure elapsed time

batch\_time.update(time.time() - end)

end = time.time()

#print('epoch= '+str(epoch))

#print('dataset length = ' + str(len(train\_dataset))+ ' batch size = '+str(args.batch\_size))

step = epoch\*len(train\_dataset)/args.batch\_size + i

#print(step)

logger.scalar\_summary('train\_loss',losses.avg,step)

#if i % args.print\_freq == 0:

# logging.info('Epoch: [{0}][{1}/{2}]'.format(epoch, i, len(train\_loader)))

# logging.info('Data time {data\_time.val:.3f} ({data\_time.avg:.3f})'.format( data\_time=data\_time))

# logging.info('Loss {loss.val:.4f} ({loss.avg:.4f})'.format(loss=losses))

#print\_string = ''

#for name, value in meter\_dict.items():

# print\_string+='{name}: {loss.val:.4f} ({loss.avg:.4f})\t'.format(name=name, loss=value)

#print(print\_string)

return losses.avg

def validate(val\_loader, model, epoch):

batch\_time = AverageMeter()

data\_time = AverageMeter()

losses = AverageMeter()

meter\_dict = {}

for name in build\_names():

meter\_dict[name] = AverageMeter()

meter\_dict['max\_ht'] = AverageMeter()

meter\_dict['min\_ht'] = AverageMeter()

meter\_dict['max\_paf'] = AverageMeter()

meter\_dict['min\_paf'] = AverageMeter()

# switch to train mode

model.eval()

nb\_found = 0

end = time.time()

print('validating')

for i, (img, heatmap\_target, heat\_mask, paf\_target, paf\_mask) in enumerate(val\_loader):

#print(img.size())

if type(img[0]) == 'str':

continue

else:

nb\_found += 1

# print(nb\_found, i)

# measure data loading time

data\_time.update(time.time() - end)

img = img.squeeze(1).cuda()

heatmap\_target = heatmap\_target.squeeze(1).cuda()

heat\_mask = heat\_mask.squeeze(1).cuda()

paf\_target = paf\_target.squeeze(1).cuda()

paf\_mask = paf\_mask.squeeze(1).cuda()

# compute output

\_,saved\_for\_loss = model(img)

total\_loss, saved\_for\_log = get\_loss(saved\_for\_loss, heatmap\_target, heat\_mask,

paf\_target, paf\_mask)

for name,\_ in meter\_dict.items():

meter\_dict[name].update(saved\_for\_log[name], img.size(0))

losses.update(total\_loss.item(), img.size(0))

# measure elapsed time

batch\_time.update(time.time() - end)

end = time.time()

#if i % args.print\_freq == 0:

# logging.info('Epoch: [{0}][{1}/{2}]'.format(epoch, i, len(val\_loader)))

# logging.info('Data time {data\_time.val:.3f} ({data\_time.avg:.3f})'.format(data\_time=data\_time))

# logging.info('Loss {loss.val:.4f} ({loss.avg:.4f})'.format(loss=losses))

#print\_string = ''

#for name, value in meter\_dict.items():

# print\_string+='{name}: {loss.val:.4f} ({loss.avg:.4f})\t'.format(name=name, loss=value)

#print(print\_string)

return losses.avg

class AverageMeter(object):

"""Computes and stores the average and current value"""

def \_\_init\_\_(self):

self.reset()

def reset(self):

self.val = 0

self.avg = 0

self.sum = 0

self.count = 0

def update(self, val, n=1):

self.val = val

self.sum += val \* n

self.count += n

self.avg = self.sum / self.count

print("Loading dataset...")

# load data

# TODO update data loaders

valid\_dataset = Pose\_Dataset(args.valid\_dir, args.num\_valid)

train\_dataset = Pose\_Dataset(args.train\_dir, args.num\_train)

valid\_loader = DataLoader(valid\_dataset,

batch\_size=args.batch\_size,

num\_workers=0)

train\_loader = DataLoader(train\_dataset,

batch\_size=args.batch\_size,

shuffle=True,

num\_workers=0)

print("{:6d} train samples".format(len(train\_dataset)))

print("{:6d} valid samples".format(len(valid\_dataset)))

# model

model = get\_model(trunk='vgg19')

#model = encoding.nn.DataParallelModel(model, device\_ids=args.gpu\_ids)

#model = torch.nn.DataParallel(model)

model = torch.nn.DataParallel(model).cuda()

# load pretrained

use\_vgg(model, args.model\_path, 'vgg19')

# Fix the VGG weights first, and then the weights will be released

for i in range(20):

for param in model.module.model0[i].parameters():

param.requires\_grad = False

trainable\_vars = [param for param in model.parameters() if param.requires\_grad]

optimizer = torch.optim.SGD(trainable\_vars, lr=args.lr,

momentum=args.momentum,

weight\_decay=args.weight\_decay,

nesterov=args.nesterov)

print("Beginning training.")

print("WARNING: only non-zero loss is stage 1, branch 2")

start\_time = time.time()

for epoch in range(5):

logging.info("Epoch {}".format(epoch))

# train for one epoch

train\_loss = train(train\_loader, model, optimizer, epoch)

# evaluate on validation set

val\_loss = validate(valid\_loader, model, epoch)

logger.scalar\_summary('Val\_loss',val\_loss,epoch)

t = (time.time()-start\_time)/60.0

print('Epoch '+str(epoch)+' took '+str(t)+ ' minutes ---')

logging.info("Train loss: {:.6f}".format(train\_loss))

logging.info("Valid loss: {:.6f}".format(val\_loss))

# Release all weights

for param in model.module.parameters():

param.requires\_grad = True

trainable\_vars = [param for param in model.parameters() if param.requires\_grad]

optimizer = torch.optim.SGD(trainable\_vars, lr=args.lr,

momentum=args.momentum,

weight\_decay=args.weight\_decay,

nesterov=args.nesterov)

lr\_scheduler = ReduceLROnPlateau(optimizer, mode='min', factor=0.8, patience=5, verbose=True, threshold=0.0001, threshold\_mode='rel', cooldown=3, min\_lr=0, eps=1e-08)

best\_val\_loss = np.inf

model\_save\_filename = './network/weight/best\_pose\_test.pth'

for epoch in range(5, args.epochs):

# train for one epoch

train\_loss = train(train\_loader, model, optimizer, epoch)

# evaluate on validation set

val\_loss = validate(valid\_loader, model, epoch)

t = (time.time()-start\_time)/60.0

print('Epoch '+str(epoch)+' took '+str(t)+ ' minutes ---')

logger.scalar\_summary('Val\_loss',val\_loss,epoch)

logging.info("Train loss: {:.6f}".format(train\_loss))

logging.info("Valid loss: {:.6f}".format(val\_loss))

lr\_scheduler.step(val\_loss)

is\_best = val\_loss<best\_val\_loss

best\_val\_loss = min(val\_loss, best\_val\_loss)

if is\_best:

torch.save(model.state\_dict(), model\_save\_filename)