import torch

from torch.nn import init

import functools

from torch.autograd import Variable

from torch.optim import lr\_scheduler

import numpy as np

from torchvision import models

import torch.nn as nn

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

# Functions

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

def weights\_init\_normal(m):

classname = m.\_\_class\_\_.\_\_name\_\_

# print(classname)

if classname.find('Conv') != -1:

init.normal\_(m.weight.data, 0.0, 0.02)

elif classname.find('Linear') != -1:

init.normal\_(m.weight.data, 0.0, 0.02)

elif classname.find('BatchNorm2d') != -1:

init.normal\_(m.weight.data, 1.0, 0.02)

init.constant(m.bias.data, 0.0)

def weights\_init\_xavier(m):

classname = m.\_\_class\_\_.\_\_name\_\_

# print(classname)

if classname.find('Conv') != -1:

init.xavier\_normal(m.weight.data, gain=0.02)

elif classname.find('Linear') != -1:

init.xavier\_normal(m.weight.data, gain=0.02)

elif classname.find('BatchNorm2d') != -1:

init.normal(m.weight.data, 1.0, 0.02)

init.constant(m.bias.data, 0.0)

def weights\_init\_kaiming(m):

classname = m.\_\_class\_\_.\_\_name\_\_

# print(classname)

if classname.find('Conv') != -1:

init.kaiming\_normal(m.weight.data, a=0, mode='fan\_in')

elif classname.find('Linear') != -1:

init.kaiming\_normal(m.weight.data, a=0, mode='fan\_in')

elif classname.find('BatchNorm2d') != -1:

init.normal(m.weight.data, 1.0, 0.02)

init.constant(m.bias.data, 0.0)

def weights\_init\_orthogonal(m):

classname = m.\_\_class\_\_.\_\_name\_\_

print(classname)

if classname.find('Conv') != -1:

init.orthogonal(m.weight.data, gain=1)

elif classname.find('Linear') != -1:

init.orthogonal(m.weight.data, gain=1)

elif classname.find('BatchNorm2d') != -1:

init.normal(m.weight.data, 1.0, 0.02)

init.constant(m.bias.data, 0.0)

def init\_weights(net, init\_type='normal'):

print('initialization method [%s]' % init\_type)

if init\_type == 'normal':

net.apply(weights\_init\_normal)

elif init\_type == 'xavier':

net.apply(weights\_init\_xavier)

elif init\_type == 'kaiming':

net.apply(weights\_init\_kaiming)

elif init\_type == 'orthogonal':

net.apply(weights\_init\_orthogonal)

else:

raise NotImplementedError('initialization method [%s] is not implemented' % init\_type)

def network\_vgg16(cpu=False):

# ImageNetで事前トレーニングされたVGG16モデルを取得

model = models.vgg16(pretrained=True)

# VGG16の出力層のユニット数を2にする

model.classifier[6] = nn.Linear(

in\_features=4096, # 入力サイズはデフォルトの4096

out\_features=2) # 出力はデフォルトの1000から2に変更

# 出力層の重みとバイアスを更新可として登録

update\_param\_names = ['classifier.6.weight', 'classifier.6.bias']

# 出力層以外は勾配計算をなくし、変化しないように設定

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

if name in update\_param\_names:

param.requires\_grad = True # 勾配計算を行う

#params\_to\_update.append(param) # パラメーター値を更新

print(name) # 更新するパラメーター名を出力

else:

param.requires\_grad = False # 出力層以外は勾配計算なし

if not cpu:

model.cuda()

return model