from torch.utils.data import Dataset

from PIL import Image

from pathlib import Path

import glob

from collections import namedtuple

import os

CandidateInfoTuple = namedtuple('CandidateInfoTuple', ['label','path'])

def make\_path\_list(root\_dir,phase='train',):

'''

データのファイルパスを格納したリストを作成する。

Parameters:

phase(str): 'train'または'val'

Returns:

path\_list(list): 画像データのパスを格納したリスト

'''

# 画像ファイルのルートディレクトリ

# 画像ファイルパスのフォーマットを作成

# rootpath +

# train/ants/\*.jpg

# train/bees/\*.jpg

# val/ants/\*.jpg

# val/bees/\*.jpg

tt= phase +'/\*\*/\*.jpg'

target\_path = os.path.join(root\_dir,tt)

# ファイルパスを格納するリスト

candidateInfo\_list = [] # ここに格納する

# glob()でファイルパスを取得してリストに追加

for path in glob.glob(target\_path):

p=Path(path)

label=p.parts[-2]

if label == 'ants':

label = 0 # アリは0

else:

label =1 #Beesは、1

candidateInfo\_list.append(

CandidateInfoTuple(

label,

path

)

)

return candidateInfo\_list

# ファイルパスのリストを生成

#train\_list = make\_datapath\_list(phase='train')

#val\_list = make\_datapath\_list(phase='val')

class MakeDataset(Dataset):

'''

アリとハチの画像のDatasetクラス

PyTorchのDatasetクラスを継承

Attributes:

file\_list(list): 画像のパスを格納したリスト

transform(object): 前処理クラスのインスタンス

phase(str): 'train'または'val'

Returns:

img\_transformed: 前処理後の画像データ

label(int): 正解ラベル

'''

def \_\_init\_\_(self, file\_list, transform=None, phase='train'):

'''インスタンス変数の初期化

'''

self.file\_list = file\_list[phase] # ファイルパスのリスト

self.transform = transform # 前処理クラスのインスタンス

self.phase = phase # 'train'または'val'

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

'''len(obj)で実行されたときにコールされる関数

画像の枚数を返す'''

return len(self.file\_list)

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

'''Datasetクラスの\_\_getitem\_\_()をオーバーライド

obj[i]のようにインデックスで指定されたときにコールバックされる

Parameters:

index(int): データのインデックス

Returns:

前処理をした画像のTensor形式のデータとラベルを取得

'''

# ファイルパスのリストからindex番目の画像をロード

img\_path = self.file\_list[index]

# ファイルを開く -> (高さ, 幅, RGB)

img = Image.open(img\_path)

# 画像を前処理 -> torch.Size([3, 224, 224])

img\_transformed = self.transform(

img, self.phase)

p=Path(img\_path)

label=p.parts[-2]

# 正解ラベルをファイル名から切り出す

#if self.phase == 'train':

# 訓練データはファイルパスの31文字から34文字が'ants'または'bees'

# p=Path(img\_path)

# label=p.parts[-2]

#elif self.phase == 'val':

# # 検証データはファイルパスの29文字から32文字が'ants'または'bees'

# label=p.parts[-2]

# 正解ラベルの文字列を数値に変更する

if label == 'ants':

label = 0 # アリは0

elif label == 'bees':

label = 1 # ハチは1

return img\_transformed, label

import random

class MakeBalancedDataset(Dataset):

'''

アリとハチの画像のDatasetクラス

PyTorchのDatasetクラスを継承

Attributes:

file\_list(list): 画像のパスを格納したリスト

transform(object): 前処理クラスのインスタンス

phase(str): 'train'または'val'

Returns:

img\_transformed: 前処理後の画像データ

label(int): 正解ラベル

'''

def \_\_init\_\_(self, file\_list, ratio\_int=True, transform=None, phase='train',records=300):

'''インスタンス変数の初期化

'''

self.file\_list = file\_list[phase] # ファイルパスのリスト

self.transform = transform # 前処理クラスのインスタンス

self.phase = phase # 'train'または'val'

self.ratio\_int= ratio\_int

self.records = records

random.shuffle(self.file\_list)

self.ants\_list = [nt for nt in self.file\_list if not nt.label]

self.bees\_list = [nt for nt in self.file\_list if nt.label]

def shuffleSamples(self):

if self.ratio\_int:

random.shuffle(self.ants\_list)

random.shuffle(self.bees\_list)

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

if self.ratio\_int:

return self.records

else:

return len(self.file\_list)

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

if self.ratio\_int:

pos\_ndx = ndx // (self.ratio\_int + 1)

#奇数のとき

if ndx % (self.ratio\_int + 1):

neg\_ndx = ndx - 1 - pos\_ndx

neg\_ndx %= len(self.ants\_list)

candidateInfo\_tup = self.ants\_list[neg\_ndx]

#偶数のとき

else:

pos\_ndx %= len(self.bees\_list)

candidateInfo\_tup = self.bees\_list[pos\_ndx]

else:

candidateInfo\_tup = self.file\_list[ndx]

# ファイルを開く -> (高さ, 幅, RGB)

img = Image.open(candidateInfo\_tup.path)

# 画像を前処理 -> torch.Size([3, 224, 224])

img\_transformed = self.transform(

img, self.phase)

return img\_transformed, candidateInfo\_tup.label