Borowicz et al. Supplement 2

January 29, 2019

1 Supplement 2

1.1 2.1 Utilities called in other scripts

```
########################################################################################
#### Utility codes and  that are called for spacewhale
#### Authors: Hieu Le, Grant Humphries, Alex Borowicz
#### Date: August 2018
########################################################################################
from __future__ import print_function, division

import os
import numpy as np
from scipy import misc
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision
from torchvision import datasets, models, transforms
import time

class spacewhale:
    def __init__(self):
        ##### These are the data transforms used throughout the code - they are called on in o
        ### These transformations convert images into tensors, which can be used by pytorch mo
        self.data_transforms = {
            'train': transforms.Compose([
                transforms.RandomRotation(10),
                transforms.RandomResizedCrop(224),
                transforms.RandomHorizontalFlip(),
                transforms.RandomVerticalFlip(),
                transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1),
                transforms.ToTensor(),
                transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
            ]),
            'val': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
        'test': transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
    }


### Create a directory if one does not exist
def sdmkdir(self,d):
    if not os.path.isdir(d):
        os.makedirs(d)

### Tile images into small pieces to feed to model
def savepatch_train(self,png,w,h,step,size,imbasename):

    ni = np.int32(np.floor((w- size)/step) +2)
    nj = np.int32(np.floor((h- size)/step) +2)

    for i in range(0,ni-1):
        for j in range(0,nj-1):
            name = format(i,'03d')+'_'+format(j,'03d')+'.png'
            misc.toimage(png[i*step:i*step+size,j*step:j*step+size,:]).save(imbasename+name
    for i in range(0,ni-1):
        name = format(i,'03d')+'_'+format(nj-1,'03d')+'.png'
        misc.toimage(png[i*step:i*step+size,h-size:h,:]).save(imbasename+format(i,'03d')+'


    for j in range(0,nj-1):
        name = format(ni-1,'03d')+'_'+format(j,'03d')+'.png'
        misc.toimage(png[w-size:w,j*step:j*step+size,:]).save(imbasename+format(ni-1,'03d'

    misc.toimage(png[w-size:w,h-size:h,:]).save(imbasename+format(ni-1,'03d')+'_'+format(n

### Training a CNN model
def train_model(self, opt, device, dataset_sizes, dataloaders, model,criterion, optimizer,

    since = time.time()

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)
```

```
            for phase in ['train']:
                if phase == 'train':
                    scheduler.step()
                    model.train()  # Set model to training mode
                    filename = 'epoch_'+str(epoch)+'.pth'
                else:
                    model.eval()   # Set model to evaluate mode

                running_loss = 0.0
                running_corrects = 0
                running_errors = 0

                tp=0
                tn=0
                fp=0
                fn=0

                # Iterate over data.
#                for inputs, labels  in dataloaders[phase]:
                for batch_index, (inputs, labels) in enumerate(dataloaders):

                    inputs = inputs.to(device)
                    labels = labels.to(device)

                    # zero the parameter gradients
                    optimizer.zero_grad()

                    # forward
                    # track history if only in train
                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)
                        # compute cross entropy loss
                        loss = criterion(outputs, labels)
                        # get prediction for the statistics
                        _, preds = torch.max(outputs, 1)

                        # backward + optimize only if in training phase
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()

                    # statistics
                    running_loss += loss.item() * inputs.size(0)
                    running_corrects += torch.sum(preds == labels.data)
                    running_errors += torch.sum(preds != labels.data)

                    tp += torch.sum(preds[labels.data==0] == 0)
                    fn += torch.sum(preds[labels.data==0] == 1)
                fp += torch.sum(preds[labels.data==1] == 0)
                tn += torch.sum(preds[labels.data==1] ==1)


            # calculate loss, accuracy, error in the model epoch
            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]
            epoch_err = running_errors.double() / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f} Err: {:.4f}'.format(
                phase, epoch_loss, epoch_acc, epoch_err))
            ### save the model
            torch.save(model.state_dict(),opt.checkpoint+'/'+filename)

            print('TP: {:.4f}  TN: {:.4f}  FP: {:.4f}  FN: {:.4f}'.format(tp, tn, fp, fn))

    time_elapsed = time.time() - since
    print('-----------------------------------------------------------')
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('-----------------------------------------------------------')


### Test the pre-loaded model on a single image
def test_im(self,device,model_ft,class_names,test_transforms,im):
    A_img = Image.open(im)
    A_img = A_img.resize((224, 224),Image.NEAREST)
    A_img = test_transforms(A_img)
    A_img = torch.unsqueeze(A_img,0)
    A_img = A_img.to(device)
    pred = model_ft(A_img)
    print(pred.max())



### Test the pre-loaded model on a chosen directory
def test_dir(self,device,model_ft,dataloader):
    tp=0
    fp=0
    tn=0
    fn=0
    for im, labs in dataloader:
        im, labs = im.to(device), labs.to(device)
        outputs = model_ft(im)
        outputs = outputs
        _,preds = torch.max(outputs,1)

        tp = tp+ torch.sum(preds[labs==0] == 0)
```

```
            fn = fn+ torch.sum(preds[labs==0] == 1)
            fp = fp +torch.sum(preds[labs==1] == 0)
            tn = tn + torch.sum(preds[labs==1] ==1)



        print('Correctly Identified as Water: '+ str(float(tp)))
        print('Correctly Identified as Whales: '+ str(float(tn)))
        print('Misidentified as Water: '+ str(float(fp)))
        print('Misidentified as Whales: '+ str(float(fn)))

        prec = float(tp)/float(tp+fp)
        recall =  float(tp)/ float(tp+fn)
        print("prec: %f, recall: %f"%(prec,recall))

    ### A weighted random sampler to deal with the lopsided size of classes
    ### Specifically fewer sat images than aerial. Adapted from
    ### https://discuss.pytorch.org/t/balanced-sampling-between-classes-with-
    ### torchvision-dataloader/2703/3
    def make_weights_for_balanced_classes(self, images, nclasses):
        count = [0] * nclasses
        for item in images:
            count[item[1]] += 1
        weight_per_class = [0.] * nclasses
        N = float(sum(count))
        for i in range(nclasses):
            weight_per_class[i] = N/float(count[i])
        weight = [0] * len(images)
        for idx, val in enumerate(images):
            weight[idx] = weight_per_class[val[1]]
        return weight
```

1.2 Supplement 2.2 Tile images for training

```
##################################################################
### This script is used to tile images into smaller images for ease of use
### within the process
### Use example:
### python gen_training_patches.py --root home/test_imagery --step 32 --size 32 --output home/t
### You can set the step size (# of pixels between the beginning of one image and the next)
### And size (the # of pixels in an image). Combined these give you how many tiles will be crea
### and the degree of overlap between them. If step and size are the same, there will be
### no overlap. If step is less than size, there will be overlap. If step is greater than
### size, there will be gaps between the tiles that are ignored.
import numpy as np
from PIL import Image
import time
import torch
import os.path
import argparse
from scipy import misc
from m_util import *


s = spacewhale()

parse = argparse.ArgumentParser()
parse.add_argument('--root',type=str,default='./Water_Training')
parse.add_argument('--step',type=int,default=500)
parse.add_argument('--size',type=int,default=30)
parse.add_argument('--output',type=str,default='./water')
opt = parse.parse_args()
opt.im_fold = opt.root
opt.results = opt.output

s.sdmkdir(opt.results)
opt.input_nc =3
imlist=[]
imnamelist=[]

for root,_,fnames in sorted(os.walk(opt.root)):
    for fname in fnames:
        if fname.lower().endswith('.png'):
            path = os.path.join(root,fname)
            imlist.append((path,fname))
            imnamelist.append(fname)

for im_path,imname in  imlist:
    png = misc.imread(im_path,mode='RGB')
    w,h,z = png.shape

    s.savepatch_train(png,w,h,opt.step,opt.size,opt.results+'/'+imname[:-4]+'#')
```

1.3 Supplement 2.3 Model training

```
##############################################################################################
#### Script for training a pytorch, convolutional neural net, using the pre-trained resnet18 mo
#### Authors:  Hieu Le, Grant Humphries, Alex Borowicz
#### Date: August 2018
#### This script was written for the Spacewhale project
#### and was based on the Pytorch transfer learning tutorial:
#### https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
##############################################################################################
#### Usage examples (Linux)
####
####  python training_script.py --name MODEL1 --data_dir /home/ghumphries/spacewhale/data --ve
```

```
####
#################################################################################################
#### Setup information
####    To run this script, ensure that you have your training images inside of a folder callec
####    Inside of the train folder, your images must be organized into folders based on the lal
####    ./train/Water - this folder contains only water images in .png format
####    ./train/Whale - this folder contains only whale images in .png format
####    IMPORTANT:
####        The --data_dir argument must point to the folder ABOVE the 'train' folder. For exan
####        .home/user/spacewhale/fulldata/train/... ->  data_dir usage:  --data_dir /home/use
####
#################################################################################################
### Library imports
from __future__ import print_function, division

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import os
from spacewhale_util import *
import argparse
#################################################################################################

### Create arguments for command line interface
parser = argparse.ArgumentParser()
parser.add_argument('--name',type=str)
parser.add_argument('--data_dir',type=str)
parser.add_argument('--verbose',type=bool,default=False)
parser.add_argument('--epochs',type=int,default=25)

opt = parser.parse_args()

### Create the spacewhale class
s = spacewhale()

### This creates a new directory called 'trained model' in the directory you are currently worl
opt.checkpoint = ('./trained_model/'+opt.name)
s.sdmkdir(opt.checkpoint)

#Preparing the data
print('###########################################################################################
print('WELCOME TO SPACEWHALE!')
print('###########################################################################################
print('We will now train your model. Please be patient')
print('----------------------------------------------------------------------------')
```

```
### This part loads up any folders in the 'train' folder with the label being the name of the f
image_datasets = {x: datasets.ImageFolder(os.path.join(opt.data_dir, x),s.data_transforms[x]) f
weights = s.make_weights_for_balanced_classes(image_datasets['train'].imgs, len(image_datasets[
weights = torch.DoubleTensor(weights)
sampler = torch.utils.data.sampler.WeightedRandomSampler(weights, len(weights))
dataloaders = torch.utils.data.DataLoader(image_datasets['train'], batch_size=4, sampler = samp
dataset_sizes = {x: len(image_datasets[x]) for x in ['train']}

print('Your dataset size is: %d'%(dataset_sizes['train']))
class_names = image_datasets['train'].classes
print('You have',str(len(class_names)),'classes in your dataset')

print('-------------------------------------------------------------------------------')
print('Labels for the dataset are:')
print(class_names[0] + ' = 0')
print(class_names[1] + ' = 1')
print('-------------------------------------------------------------------------------')
### This sets the device (if cuda is installed properly, it will send the training data to the
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
dev = ("gpu" if torch.cuda.is_available() else "cpu")
print('Data loaded into', dev)
print('-------------------------------------------------------------------------------')

################################################################################################
### This part defines the model we're going to use
### First it downloads the pretrained resnet model (if we dont' have it) from modelZoo
### We count the number of features in the model and then replace the last layer with a linear
### The model is sent to the gpu and we then opt to use CrossEntropy as the loss function
### The optimizer is set as stochastic gradient descent with a learning rate of 0.001
### We then set the learning rate to decay every 7 epochs

model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
model_ft = model_ft.to(device)
criterion = nn.CrossEntropyLoss()
# Set Learning rate (lr) and step size below
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.0009, momentum=0.9)
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

### If the verbose option is set, then print out the model
if opt.verbose:
    print(model_ft)

################################################################################################
### Run the train_model function from the spacewhale class
```

```
model_ft = s.train_model(opt, device, dataset_sizes, dataloaders, model_ft, criterion, optimize

##############################################################################################
```

1.4 Supplement 2.4 Model testing

```
##############################################################################################
#### Script for testing a pytorch, convolutional neural net, using the pre-trained resnet18 mo
#### Authors:  Hieu Le & Grant Humphries
#### Date: August 2018
#### This script was written for the Spacewhale project
#### and was based on the Pytorch transfer learning tutorial:
#### https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
##############################################################################################
#### Usage examples (Linux)
####
####  python testing_script.py --data_dir /home/ghumphries/spacewhale/test --model MODEL1 --ep
####
##############################################################################################
#### Setup information
####    To run this script, ensure that you have folders named exactly the same as those in th
####    For example:
####    ./test/Water
####    ./test/Whale
####    IMPORTANT:
####        The images that you want to test should all live in the target folder. For example
####        water, then place all the images in the ./test/Water folder. If you want to test f
####        the ./test/Whale folder
####        The data_dir argument should point to the directory ABOVE the training folders.
####        For example, if your directory is:  /home/user/spacewhale/testingdata/Water
####        then --data_dir /home/user/spacewhale/testingdata
##############################################################################################
### Library imports

from __future__ import print_function, division
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models, transforms
from spacewhale_util import *
import os
import argparse


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

### Create arguments for command line interface
```

```
parse = argparse.ArgumentParser()
parse.add_argument('--data_dir')
parse.add_argument('--model')
parse.add_argument('--epoch',type=int,default=24)
opt = parse.parse_args()

### Create the spacewhale class
s = spacewhale()

### Specify which epoch to load from the pre-trained model
epoch_to_use = 'epoch_'+str(opt.epoch)+'.pth'
trained_model = os.path.join('./trained_model',opt.model,epoch_to_use)

### Transform image to tensor for testing
test_transforms = s.data_transforms['test']

### Load the model into GPU:
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
torch.set_default_tensor_type('torch.cuda.FloatTensor')
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
model_ft = model_ft.to(device)
model_ft.load_state_dict(torch.load(trained_model))

### Set the model into evaluation mode:
model_ft.eval()

## Data loader for the testing dataset
image_datasets = datasets.ImageFolder(opt.data_dir, s.data_transforms['test'])
dataloaders = torch.utils.data.DataLoader(image_datasets, batch_size=10,shuffle=False, num_worl

### Run the model to predict all images in a directory
s.test_dir(device,model_ft,dataloaders)
```