

# Supplement S2 File

January 29, 2019

## 1 Supplement S2 File

### 1.1 S2 File. Utilities called in other scripts.

```
In [ ]: #####  
        #### 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 other scripts. These transformations convert images  
                ### into tensors, which can be used by pytorch models, and apply data  
                ### augmentation methods  
                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')+'_'+
                                                format(nj-1,'03d')+'.png')

    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')+

```

```

                                '_' + format(j, '03d') + '.png')

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

### Training a CNN model
def train_model(self, opt, device, dataset_sizes, dataloaders, model, criterion,
                optimizer, scheduler, num_epochs=25):

    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

```