Codes for the 1st Project in NAOC

Yao Zhang

1 Data Match

1.1 resize_fits.m

```
clear;clc;
%filesep;
root_path = pwd;
dir_fits_resize = fullfile(root_path,'fits','fits_data_resize');
if ~exist(dir_fits_resize,'dir')
    mkdir (dir_fits_resize)
end
dir_fits_folders = fullfile(root_path,'fits_orig');
cd (dir_fits_folders);
dir_folders = dir(dir_fits_folders);
dir_folders = dir_folders(~ismember({dir_folders.name},{'.','..'}));
[num_folder_fits,~] = size(dir_folders);
cd (dir_fits_resize);
for k = 1:num_folder_fits
    sub_folders_name = dir_folders(k).name;
    if ~exist(sub_folders_name,'dir')
    mkdir (sub_folders_name)
    end
end
cd (dir_fits_folders);
for  i = 1:num_folder_fits
    cd (dir_folders(i).name)
    fileExt = '*.fits';
    fnames = dir(fullfile(pwd,fileExt));
    len = size(fnames,1);
    for j = 1:len
        fits_name = fnames(j,1).name;
        len_name = length(fits_name);
        mark_end_name = fits_name(end-12:end);
        if strcmp(mark_end_name ,'overview.fits')==0
            fits_data = fitsread(fnames(j,1).name);
            info = fitsinfo(fnames(j,1).name);
            tem_info = info.PrimaryData.Keywords(:,1,:);
            [bool, index] = ismember('SOLAR_P',tem_info);
            sloar_p_num = cell2mat(info.PrimaryData.Keywords(index
    ,2));
            sloar_p_num = round(sloar_p_num);
            if sloar_p_num == -180
                fits_data = rot90(fits_data,2);
            elseif sloar_p_num == 180
                fits_data = rot90(fits_data,2);
            end
```

```
            fits_resize_data = imresize(fits_data,[1024 1024]);
            sub_folders_name = dir_folders(i).name;
            file_name = [fnames(j,1).name(1:end-5),'.mat'];
            save_path = fullfile(dir_fits_resize,sub_folders_name,
    file_name);
            save(save_path,'fits_resize_data')
        end
    end
    cd ..
end
cd ..
```

1.2 sav2txt.py

```
import os
import sys
import imageio
from scipy.io import readsav
import numpy as np
path =  sys.path[0]
dirs = os.listdir(path)
for i in dirs:
  if os.path.splitext(i)[1] == '.sav':
          savData = readsav(os.path.join(path,i))
          x = savData.imag
          y = os.path.splitext(i)[0] + '.txt'
          y = os.path.join(path,y)
          np.savetxt(y,x)
```

1.3 resize_sav.m

```
clear; clc;
root_path = pwd;
sav_dir = fullfile(root_path,'sav_orig' );
sav_resize_dir = fullfile(root_path,'sav','sav_data_resize' );
if ~exist(sav_resize_dir,'dir')
    mkdir (sav_resize_dir)
end
fileExt = '*.txt';
fnames = dir(fullfile(sav_dir,fileExt));
len = size(fnames,1);
cd (sav_dir)
for i = 1:len
    sav_data=importdata(fnames(i,1).name);
    tem_name = fnames(i,1).name;
    tem_name = tem_name(1:end-4);
    new_name = strcat(tem_name,'.mat');
```

```
    sav_resize_data = imresize(sav_data,[1024 1024]);
    sav_save_path_name = fullfile(sav_resize_dir,new_name);
    save(sav_save_path_name,'sav_resize_data')
end
cd ..
```

1.4 match_data.m

```
clear; clc;
root_path = pwd;
dir_sav_resize = fullfile(root_path,'sav','sav_data_resize');
dir_fits_resize = fullfile(root_path,'fits','fits_data_resize');
dir_final_fits_data =  fullfile(root_path,'fits','15');
if ~exist(dir_final_fits_data,'dir')
    mkdir (dir_final_fits_data)
end
cd (dir_sav_resize);
dir_sav_folders = dir(dir_sav_resize);
dir_sav_folders = dir_sav_folders(~ismember({dir_sav_folders .name
    },{'.','..'}));
[num_folder_sav,~] = size(dir_sav_folders);
for i = 1 : num_folder_sav
    tem1 = dir_sav_folders(i).name;
    sav_data_struct = load(tem1);
    sav_data = sav_data_struct.sav_resize_data;
    sav_no = tem1(end-8:end-4);
    sav_no_double = str2double(sav_no);
    nums_days_sav = sav_no_double + 684594;
    flag_subdir_fits = nums_days_sav - 727930;
    cd (dir_fits_resize);
    flag_subdir_fits = num2str(flag_subdir_fits);
    dir_subdir_fits = [dir_fits_resize,filesep,'fd_M_96m_01d.00',
    flag_subdir_fits];
    if exist(dir_subdir_fits,'dir')
       cd(dir_subdir_fits)
       sub_sub_fits_dir = dir;
       name_sub_sub = sub_sub_fits_dir(~ismember({sub_sub_fits_dir.
    name},{'.','..'}));
       len_sub_sub = length(name_sub_sub);
       tem_coef = -2* ones(1,len_sub_sub);
       for h = 1 : len_sub_sub
           tem2 = name_sub_sub(h).name;
           fits_data_struct = load(tem2);
           fits_data = fits_data_struct.fits_resize_data;
           fits_data_mask = fits_data;
           fits_data_mask(isnan(fits_data_mask)) = 0;
```

```
            fits_data_mask(fits_data_mask >= 500) = 500;
            fits_data_mask(fits_data_mask <= -500) = -500;
            corrcoef_matrix = corrcoef(sav_data,fits_data_mask);
            corrcoef_scalar = corrcoef_matrix(1,2);
            tem_coef(h) = corrcoef_scalar;
        end
        if max(tem_coef) > 0.15 & max(tem_coef) < 2
            ith = find(tem_coef == max(tem_coef));
            simlar_fits_name = name_sub_sub(ith).name;
            copyfile(simlar_fits_name,dir_final_fits_data );
        end

    end
    cd (dir_sav_resize);
end
cd ..
```

1.5 matchFitsToTxt.m

```
clear;clc;
root_path = pwd;
name_mask = '075';
dir_fits = fullfile(root_path,'fits',name_mask);
fext = '*.mat';
filearray = dir([dir_fits filesep fext]);
NumImages = size(filearray,1); % get the number of images
root_path = pwd;
if NumImages < 0
    error('No files in the directory');
end
txt_name = [name_mask '.txt'];
for i=1:NumImages
    fid = fopen(txt_name,'a+');
    newname = filearray(i).name;
    no = newname(1:end-4);
    fprintf(fid,'%s \n',no);
    fclose(fid);
end
```

1.6 move_fits_orig1st.m

```
clc; clear;
name_mask = '15';
txt_name = [name_mask '.txt'];
fid = fopen(txt_name);
allText = textscan(fid,'%s','delimiter','\n');
numberOfLines = length(allText{1});
```

```
fclose(fid);
line = numberOfLines;
dir_root = pwd;

filein =fullfile(dir_root,'fits_orig');
fileout =fullfile(dir_root,'final_fits_orig',name_mask);
if ~exist(fileout,'dir')
    mkdir (fileout)
end
fidin=fopen(filein,'r');
fidout=fopen(fileout,'w');
for i=1:line
    fits_name_patt1 = allText{1,1}{i,1}(1:end-10);
    fits_name_patt2 = '00';
    fits_name_patt3 = allText{1,1}{i,1}(end-9:end-6);
    fits_name = [fits_name_patt1,fits_name_patt2,fits_name_patt3];
    fits_sub_name = [allText{1,1}{i,1}(1:end-1), '.fits'];
    fits_sub_dir = fullfile(filein, fits_name,fits_sub_name);
    copyfile( fits_sub_dir,fileout);
end
```

1.7 move_sav1st.m

```
clc; clear;
name_mask = '15';
txt_name = [name_mask '.txt'];
fid = fopen(txt_name);
allText = textscan(fid,'%s','delimiter','n');
numberOfLines = length(allText{1});
fclose(fid);
line = numberOfLines;
dir_root = pwd;
filein =fullfile(dir_root,'sav_orig');
fileout =fullfile(dir_root,'final_sav_data',name_mask);
if ~exist(fileout,'dir')
    mkdir (fileout)
end
fidin=fopen(filein,'r');
fidout=fopen(fileout,'w');
for i=1:line
    fits_no_str = allText{1,1}{i,1}(end-9:end-6);
    fits_no = str2double( fits_no_str);
    num_days_ac = fits_no + 727930;
    sav_no = num_days_ac - 684594;
    sav_no_str = num2str(sav_no);
    sav_name = ['Magnetogram.prjt.',sav_no_str,'.txt'];
```

```
    sav_name_dir = fullfile(filein,sav_name);
    copyfile(sav_name_dir,fileout);
end
```

1.8 fits4img.m

```
dbstop if error
warning off all
clear; clc;
root_path = pwd;
name_mask = '075';
fits_dir = fullfile(root_path,'final_fits_orig',name_mask);
fits_img_dir = fullfile(root_path,'fits','fits_img', name_mask);
if ~exist(fits_img_dir,'dir')
    mkdir (fits_img_dir)
end
fileExt = '*.fits';
fnames = dir(fullfile(fits_dir,fileExt));
len = size(fnames,1);
cd (fits_dir)
for i = 1:len
    fits_name = fnames(i,1).name;
    fits_data = fitsread(fits_name);
    info = fitsinfo(fits_name);
    tem_info = info.PrimaryData.Keywords(:,1,:);
    [bool, index] = ismember('SOLAR_P',tem_info);
    sloar_p_num = cell2mat(info.PrimaryData.Keywords(index,2));
    sloar_p_num = round(sloar_p_num);
    if sloar_p_num == -180
        fits_data = rot90(fits_data,2);
    elseif sloar_p_num == 180
        fits_data = rot90(fits_data,2);
    end
    flag_fits_name = fits_name(end-13:end-10);
    flag_num = str2double(flag_fits_name);
    num_ac = flag_num + 727930;
    fits_date = datestr(num_ac);
    fits_date(fits_date=='-') = '_';
    fits_data(fits_data>=100) = 500;
    fits_data(fits_data<=-100) = -500;
    min_value = min(min(fits_data));
    max_value = max(max(fits_data));
    fits_data(isnan(fits_data)) = min_value;
    range = max_value - min_value;
    fits_data = 255/range*fits_data + 255 - 255*max_value/range;
    name_tmp = [fits_date,'.png'];
```

```
    fits_img_subdir = fullfile(fits_img_dir, name_tmp);
    imwrite(uint8(fits_data),fits_img_subdir);
end
cd (root_path)
```

1.9 sav4img.m

```
warning off all
clear; clc;
root_path = pwd;
name_mask = '15';
sav_org_dir = fullfile(root_path,'final_sav_data',name_mask );
sav_img_dir = fullfile(root_path,'sav','sav_img',name_mask );
if ~exist(sav_img_dir,'dir')
    mkdir (sav_img_dir)
end
fileExt = '*.txt';
fnames = dir(fullfile(sav_org_dir,fileExt));
len = size(fnames,1);
cd (sav_org_dir)
for i = 1:len
    sav_name = fnames(i,1).name;
    sav_data =importdata(fnames(i,1).name);
    flag_sav_name = sav_name(end-8:end-4);
    flag_num = str2double(flag_sav_name);
    num_ac = flag_num + 684594;
    sav_date = datestr(num_ac);
    sav_date(sav_date=='-') = '_';
    sav_data(sav_data >= 150) = 150;
    sav_data(sav_data <= -150) = -150;
    min_value = min(min(sav_data));
    max_value = max(max(sav_data));
    range = max_value - min_value;
    sav_data = 255/range*sav_data + 255 - 255*max_value/range;
    sav_data = imresize(sav_data,[1024 1024]);
    name_tmp = [sav_date,'.png'];
    sav_img_subdir = fullfile(sav_img_dir, name_tmp);
    imwrite(uint8(sav_data),sav_img_subdir);
end
cd (root_path)
```

1.10 img_fits_txt.m

```
clear;
dir_root = pwd;
name_mask = '075';
dir_fits_img =fullfile(dir_root,'fits','fits_img',name_mask) ;
```

```
cd(dir_fits_img)
img_dir = dir('*.png');
num_img = size(img_dir,1);
txt_name = [name_mask '_2nd' '.txt'];
for i = 1:num_img
    fid = fopen(txt_name,'a+');
    newname = img_dir(i).name;
    no = newname;
    fprintf(fid,'%s \n',no);
    fclose(fid);
end
movefile(txt_name,dir_root);
cd (dir_root)
```

1.11 move_final_sav_img.m

```
dbstop if error
clc; clear;
name_mask = '075';
txt_name = [name_mask '_2nd' '.txt'];
fid = fopen(txt_name);
allText = textscan(fid,'%s','delimiter','\n');
numberOfLines = length(allText{1});
fclose(fid);
line = numberOfLines;
dir_root = pwd;
filein =fullfile(dir_root,'sav','sav_img',name_mask);
fileout =fullfile(dir_root,'sav','sav_img_final',name_mask);
if ~exist(fileout,'dir')
    mkdir (fileout)
end
fidin=fopen(filein,'r');
fidout=fopen(fileout,'w');
for i=1:line
    sav_name = allText{1,1}{i,1}(1:end-1);
    sav_sub_dir = fullfile(filein, sav_name);
    copyfile( sav_sub_dir,fileout);
end
```

1.12 creat_AB.m

```
clc; clear;
dir_root = pwd;
name_mask = '075';
dir_fits = fullfile(dir_root, 'fits','fits_img',name_mask);
dir_sav = fullfile(dir_root, 'sav','sav_img_final',name_mask);

```

```
dir_dataset = fullfile(dir_root, 'dataset',name_mask);
if ~exist(dir_dataset,'dir')
    mkdir (dir_dataset)
end
fileExt = '*.png';
fnames = dir(fullfile(dir_fits,fileExt));
len = size(fnames,1);
for i=1:len
    a_name = fullfile(dir_fits, fnames(i).name);
    a = imread(a_name);
    b_name = fullfile(dir_sav, fnames(i).name);
    b = imread(b_name);
    dir_AB = fullfile(dir_dataset, fnames(i).name);
    imwrite(uint8([a,b]),dir_AB);
end
```

1.13 creat_AB_col.m

```
clc; clear;
dir_root = pwd;
name_mask = '2';
dir_fits = fullfile(dir_root, 'fits','fits_img',name_mask);
dir_sav = fullfile(dir_root, 'sav','sav_img_final',name_mask);

dir_dataset = fullfile(dir_root, 'dataset','col',name_mask);
if ~exist(dir_dataset,'dir')
    mkdir (dir_dataset)
end
fileExt = '*.png';
fnames = dir(fullfile(dir_fits,fileExt));
len = size(fnames,1);
for i=1:len
    a_name = fullfile(dir_fits, fnames(i).name);
    a = imread(a_name);
    b_name = fullfile(dir_sav, fnames(i).name);
    b = imread(b_name);
    dir_AB = fullfile(dir_dataset, fnames(i).name);
    imwrite(uint8([a;b]),dir_AB);
end
```

1.14 datasetsplit.m

```
clc; clear;
dir_root = pwd;
name_mask = '2';
dir_dataset = fullfile(dir_root,'dataset',name_mask);
dir_train = fullfile(dir_dataset,'train');
```

```
if ~exist(dir_train,'dir')
    mkdir (dir_train)
end
dir_test = fullfile(dir_dataset,'test');
if ~exist(dir_test,'dir')
    mkdir (dir_test)
end
dir_val = fullfile(dir_dataset,'val');
if ~exist(dir_val,'dir')
    mkdir (dir_val)
end
fileExt = '*.png';
fnames = dir(fullfile(dir_dataset,fileExt));
len = size(fnames,1);
train_num = round(0.6*len);
val_num = round(0.2*len);
mark = randperm(len);
train_mark = mark(1:train_num);
val_mark = mark(train_num+1:train_num+val_num);
for i = 1:len
    file_name = fnames(i,1).name;
    if (ismember(i,train_mark))
        file_dir = fullfile(dir_dataset,file_name);
        copyfile(file_dir,dir_train);
    elseif(ismember(i,val_mark))
        file_dir = fullfile(dir_dataset,file_name);
        copyfile(file_dir,dir_val);
    else
        file_dir = fullfile(dir_dataset,file_name);
        copyfile(file_dir,dir_test);
    end

end
```

2 pix2pix

2.1 pix2pix.py

```
import argparse
import os
import numpy as np
import math
import itertools
import time
import datetime
import sys
```

```
9
10 import torchvision.transforms as transforms
11 from torchvision.utils import save_image
12
13 from torch.utils.data import DataLoader
14 from torchvision import datasets
15 from torch.autograd import Variable
16
17 from models import *
18 from datasets import *
19
20 import torch.nn as nn
21 import torch.nn.functional as F
22 import torch
23
24 parser = argparse.ArgumentParser()
25 parser.add_argument('--epoch', type=int, default=0, help='epoch to
     start training from')
26 parser.add_argument('--n_epochs', type=int, default=200, help='
     number of epochs of training')
27 parser.add_argument('--dataset_name', type=str, default="facades",
     help='name of the dataset')
28 parser.add_argument('--batch_size', type=int, default=1, help='size
     of the batches')
29 parser.add_argument('--lr', type=float, default=0.0002, help='adam:
     learning rate')
30 parser.add_argument('--b1', type=float, default=0.5, help='adam:
     decay of first order momentum of gradient')
31 parser.add_argument('--b2', type=float, default=0.999, help='adam:
     decay of first order momentum of gradient')
32 parser.add_argument('--decay_epoch', type=int, default=100, help='
     epoch from which to start lr decay')
33 parser.add_argument('--n_cpu', type=int, default=8, help='number of
     cpu threads to use during batch generation')
34 parser.add_argument('--img_height', type=int, default=256, help='
     size of image height')
35 parser.add_argument('--img_width', type=int, default=256, help='size
      of image width')
36 parser.add_argument('--channels', type=int, default=1, help='number
     of image channels')
37 parser.add_argument('--sample_interval', type=int, default=500, help
     ='interval between sampling of images from generators')
38 parser.add_argument('--checkpoint_interval', type=int, default = 3 ,
      help='interval between model checkpoints')
39 opt = parser.parse_args()
```

```
print(opt)

os.makedirs('images/%s' % opt.dataset_name, exist_ok=True)
os.makedirs('saved_models/%s' % opt.dataset_name, exist_ok=True)

cuda = True if torch.cuda.is_available() else False

# Loss functions
criterion_GAN = torch.nn.MSELoss()
criterion_pixelwise = torch.nn.L1Loss()

# Loss weight of L1 pixel-wise loss between translated image and real image
lambda_pixel = 100

# Calculate output of image discriminator (PatchGAN)
patch = (1, opt.img_height//2**4, opt.img_width//2**4)

# Initialize generator and discriminator
generator = GeneratorUNet()
discriminator = Discriminator()

if cuda:
    generator = generator.cuda()
    discriminator = discriminator.cuda()
    criterion_GAN.cuda()
    criterion_pixelwise.cuda()

if opt.epoch != 0:
    # Load pretrained models
    generator.load_state_dict(torch.load('saved_models/%s/generator_%d.pth' % (opt.dataset_name, opt.epoch)))
    discriminator.load_state_dict(torch.load('saved_models/%s/discriminator_%d.pth' % (opt.dataset_name, opt.epoch)))
else:
    # Initialize weights
    generator.apply(weights_init_normal)
    discriminator.apply(weights_init_normal)

# Optimizers
optimizer_G = torch.optim.Adam(generator.parameters(), lr=opt.lr, betas=(opt.b1, opt.b2))
optimizer_D = torch.optim.Adam(discriminator.parameters(), lr=opt.lr, betas=(opt.b1, opt.b2))

```

```
# Configure dataloaders
transforms_ = [ transforms.Resize((opt.img_height, opt.img_width), Image.BICUBIC),
                transforms.ToTensor(),
                transforms.Normalize([0.5], [0.5]) ]

dataloader = DataLoader(ImageDataset("../../data/%s" % opt.dataset_name, transforms_=transforms_),
                        batch_size=opt.batch_size, shuffle=True, num_workers=opt.n_cpu)

val_dataloader = DataLoader(ImageDataset("../../data/%s" % opt.dataset_name, transforms_=transforms_, mode='val'),
                            batch_size=10, shuffle=True, num_workers=1)

# Tensor type
Tensor = torch.cuda.FloatTensor if cuda else torch.FloatTensor

def sample_images(batches_done):
    """Saves a generated sample from the validation set"""
    imgs = next(iter(val_dataloader))
    real_A = Variable(imgs['B'].type(Tensor))
    real_B = Variable(imgs['A'].type(Tensor))
    fake_B = generator(real_A)
    img_sample = torch.cat((real_A.data, fake_B.data, real_B.data), -2)
    save_image(img_sample, 'images/%s/%s.png' % (opt.dataset_name, batches_done), nrow=5, normalize=True)

# ----------
#  Training
# ----------

prev_time = time.time()

for epoch in range(opt.epoch, opt.n_epochs):
    for i, batch in enumerate(dataloader):

        # Model inputs
        real_A = Variable(batch['B'].type(Tensor))
        real_B = Variable(batch['A'].type(Tensor))

        # Adversarial ground truths
        valid = Variable(Tensor(np.ones((real_A.size(0), *patch))),
```

```
    requires_grad=False)
118         fake = Variable(Tensor(np.zeros((real_A.size(0), *patch))),
    requires_grad=False)
119
120         # ------------------
121         #  Train Generators
122         # ------------------
123
124         optimizer_G.zero_grad()
125
126         # GAN loss
127         fake_B = generator(real_A)
128         pred_fake = discriminator(fake_B, real_A)
129         loss_GAN = criterion_GAN(pred_fake, valid)
130         # Pixel-wise loss
131         loss_pixel = criterion_pixelwise(fake_B, real_B)
132
133         # Total loss
134         loss_G = loss_GAN + lambda_pixel * loss_pixel
135
136         loss_G.backward()
137
138         optimizer_G.step()
139
140         # ---------------------
141         #  Train Discriminator
142         # ---------------------
143
144         optimizer_D.zero_grad()
145
146         # Real loss
147         pred_real = discriminator(real_B, real_A)
148         loss_real = criterion_GAN(pred_real, valid)
149
150         # Fake loss
151         pred_fake = discriminator(fake_B.detach(), real_A)
152         loss_fake = criterion_GAN(pred_fake, fake)
153
154         # Total loss
155         loss_D = 0.5 * (loss_real + loss_fake)
156
157         loss_D.backward()
158         optimizer_D.step()
159
160         # --------------
```

```
161         #  Log Progress
162         # --------------
163 
164         # Determine approximate time left
165         batches_done = epoch * len(dataloader) + i
166         batches_left = opt.n_epochs * len(dataloader) - batches_done
167         time_left = datetime.timedelta(seconds=batches_left * (time.
    time() - prev_time))
168         prev_time = time.time()
169 
170         # Print log
171         sys.stdout.write("\r[Epoch %d/%d] [Batch %d/%d] [D loss: %f]
     [G loss: %f, pixel: %f, adv: %f] ETA: %s" %
172                                                         (epoch, opt.
    n_epochs,
173                                                         i, len(
    dataloader),
174                                                         loss_D.item
    (), loss_G.item(),
175                                                         loss_pixel.
    item(), loss_GAN.item(),
176                                                         time_left))
177 
178         # If at sample interval save image
179         if batches_done % opt.sample_interval == 0:
180             sample_images(batches_done)
181 
182 
183     if opt.checkpoint_interval != -1 and epoch % opt.
    checkpoint_interval == 0:
184         # Save model checkpoints
185         torch.save(generator.state_dict(), 'saved_models/%s/
    generator_%d.pth' % (opt.dataset_name, epoch))
186         torch.save(discriminator.state_dict(), 'saved_models/%s/
    discriminator_%d.pth' % (opt.dataset_name, epoch))
```

2.2 models.py

```
1 import torch.nn as nn
2 import torch.nn.functional as F
3 import torch
4 
5 def weights_init_normal(m):
6     classname = m.__class__.__name__
7     if classname.find('Conv') != -1:
8         torch.nn.init.normal_(m.weight.data, 0.0, 0.02)
```

```
    elif classname.find('BatchNorm2d') != -1:
        torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
        torch.nn.init.constant_(m.bias.data, 0.0)

##############################
#           U-NET
##############################

class UNetDown(nn.Module):
    def __init__(self, in_size, out_size, normalize=True, dropout=0.0):
        super(UNetDown, self).__init__()
        layers = [nn.Conv2d(in_size, out_size, 4, 2, 1, bias=False)]
        if normalize:
            layers.append(nn.InstanceNorm2d(out_size))
        layers.append(nn.LeakyReLU(0.2))
        if dropout:
            layers.append(nn.Dropout(dropout))
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)

class UNetUp(nn.Module):
    def __init__(self, in_size, out_size, dropout=0.0):
        super(UNetUp, self).__init__()
        layers = [  nn.ConvTranspose2d(in_size, out_size, 4, 2, 1, bias=False),
                    nn.InstanceNorm2d(out_size),
                    nn.ReLU(inplace=True)]
        if dropout:
            layers.append(nn.Dropout(dropout))

        self.model = nn.Sequential(*layers)

    def forward(self, x, skip_input):
        x = self.model(x)
        x = torch.cat((x, skip_input), 1)

        return x

class GeneratorUNet(nn.Module):
    def __init__(self, in_channels=1, out_channels=1):
        super(GeneratorUNet, self).__init__()

```

```
52          self.down1 = UNetDown(in_channels, 64, normalize=False)
53          self.down2 = UNetDown(64, 128)
54          self.down3 = UNetDown(128, 256)
55          self.down4 = UNetDown(256, 512, dropout=0.5)
56          self.down5 = UNetDown(512, 512, dropout=0.5)
57          self.down6 = UNetDown(512, 512, dropout=0.5)
58          self.down7 = UNetDown(512, 512, dropout=0.5)
59          self.down8 = UNetDown(512, 512, normalize=False, dropout
    =0.5)
60
61          self.up1 = UNetUp(512, 512, dropout=0.5)
62          self.up2 = UNetUp(1024, 512, dropout=0.5)
63          self.up3 = UNetUp(1024, 512, dropout=0.5)
64          self.up4 = UNetUp(1024, 512, dropout=0.5)
65          self.up5 = UNetUp(1024, 256)
66          self.up6 = UNetUp(512, 128)
67          self.up7 = UNetUp(256, 64)
68
69
70          self.final = nn.Sequential(
71              nn.Upsample(scale_factor=2),
72              nn.ZeroPad2d((1, 0, 1, 0)),
73              nn.Conv2d(128, out_channels, 4, padding=1),
74              nn.Tanh()
75          )
76
77
78      def forward(self, x):
79          # U-Net generator with skip connections from encoder to
    decoder
80          d1 = self.down1(x)
81          d2 = self.down2(d1)
82          d3 = self.down3(d2)
83          d4 = self.down4(d3)
84          d5 = self.down5(d4)
85          d6 = self.down6(d5)
86          d7 = self.down7(d6)
87          d8 = self.down8(d7)
88          u1 = self.up1(d8, d7)
89          u2 = self.up2(u1, d6)
90          u3 = self.up3(u2, d5)
91          u4 = self.up4(u3, d4)
92          u5 = self.up5(u4, d3)
93          u6 = self.up6(u5, d2)
94          u7 = self.up7(u6, d1)
```

```
95
96          return self.final(u7)
97
98
99  ##############################
100 #         Discriminator
101 ##############################
102
103 class Discriminator(nn.Module):
104     def __init__(self, in_channels=1):
105         super(Discriminator, self).__init__()
106
107         def discriminator_block(in_filters, out_filters,
    normalization=True):
108             """Returns downsampling layers of each discriminator
    block"""
109             layers = [nn.Conv2d(in_filters, out_filters, 4, stride
    =2, padding=1)]
110             if normalization:
111                 layers.append(nn.InstanceNorm2d(out_filters))
112             layers.append(nn.LeakyReLU(0.2, inplace=True))
113             return layers
114
115         self.model = nn.Sequential(
116             *discriminator_block(in_channels*2, 64, normalization=
    False),
117             *discriminator_block(64, 128),
118             *discriminator_block(128, 256),
119             *discriminator_block(256, 512),
120             nn.ZeroPad2d((1, 0, 1, 0)),
121             nn.Conv2d(512, 1, 4, padding=1, bias=False)
122         )
123
124     def forward(self, img_A, img_B):
125         # Concatenate image and condition image by channels to
    produce input
126         img_input = torch.cat((img_A, img_B), 1)
127         return self.model(img_input)
```

2.3 datasets.py

```
1 import glob
2 import random
3 import os
4 import numpy as np
5
```

```
from torch.utils.data import Dataset
from PIL import Image
import torchvision.transforms as transforms

class ImageDataset(Dataset):
    def __init__(self, root, transforms_=None, mode='train'):
        self.transform = transforms.Compose(transforms_)

        self.files = sorted(glob.glob(os.path.join(root, mode) + '/*.*'))
        if mode == 'train':
            self.files.extend(sorted(glob.glob(os.path.join(root, 'test') + '/*.*')))

    def __getitem__(self, index):

        file = self.files[index % len(self.files)]
        img = Image.open(file)
        w, h = img.size
        img_A = img.crop((0, 0, w/2, h))
        #img_A = img_A.convert('L')
        img_B = img.crop((w/2, 0, w, h))
        #img_B = img_B.convert('L')

        if np.random.random() < 0.5:
            img_A = Image.fromarray(np.array(img_A)[::-1])
            img_B = Image.fromarray(np.array(img_B)[::-1])

        img_A = self.transform(img_A)
        img_B = self.transform(img_B)
        #img_C = img_A
        #img_A = img_B
        #img_B = img_C

        return {'A': img_A, 'B': img_B}

    def __len__(self):
        return len(self.files)


class TestDataset(Dataset):
    def __init__(self, root, transforms_=None, mode='test'):
        self.transform = transforms.Compose(transforms_)

        self.files = sorted(glob.glob(os.path.join(root, mode) + '
```

```
    /*.*'))
        if mode == 'test':
            self.files.extend(sorted(glob.glob(os.path.join(root, '
    test') + '/*.*')))

    def __getitem__(self, index):

        img = Image.open(self.files[index % len(self.files)])
        w, h = img.size
        img_A = img.crop((0, 0, w/2, h))
        #img_A = img_A.convert('RGB')
        img_B = img.crop((w/2, 0, w, h))
        #img_B = img_B.convert('RGB')

        img_A = self.transform(img_A)
        img_B = self.transform(img_B)

        return {'A': img_A, 'B': img_B}

    def __len__(self):
        return len(self.files)
```

2.4 test.py

```
import argparse
import os
import numpy as np
import math
import itertools
import time
import datetime
import sys

import torchvision.transforms as transforms
from torchvision.utils import save_image

from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

from models import *
from datasets import *

import torch.nn as nn
import torch.nn.functional as F
import torch
```

```
from PIL import Image

parser = argparse.ArgumentParser()
#parser.add_argument('--epoch', type=int, default=25, help='epoch to start training from')
parser.add_argument('--dataset_name', type=str, default="ours_500", help='name of the dataset')
parser.add_argument('--batch_size', type=int, default=1, help='size of the batches')
parser.add_argument('--img_height', type=int, default=256, help='size of image height')
parser.add_argument('--img_width', type=int, default=256, help='size of image width')
parser.add_argument('--channels', type=int, default=1, help='number of image channels')
#parser.add_argument('--checkpoint_interval', type=int, default=-1, help='interval between model checkpoints')
opt = parser.parse_args()
print(opt)

transforms_ = [ transforms.Resize((opt.img_height, opt.img_width), Image.BICUBIC),
                transforms.ToTensor(),
                transforms.Normalize([0.5], [0.5]) ]

transform_image = transforms.Compose(transforms_)
patch = (1, opt.img_height//2**4, opt.img_width//2**4)
generator = GeneratorUNet()

def get_test_img(file,transform_image):
  img = Image.open('/home/huangx/disk4T/zhangy/G2/GAN2/data/ours_500/test/' + file)
  w, h = img.size
  img_A = img.crop((0, 0, w/2, h))
  #img_A = img_A.convert('RGB')
  img_B = img.crop((w/2, 0, w, h))
  #img_B = img_B.convert('RGB')
  img_A = transform_image(img_A)
  img_B = transform_image(img_B)

  img_A = img_A.view(-1, 1, 256, 256)
  img_B = img_B.view(-1, 1, 256, 256)

  return {'img_A': img_A, 'img_B': img_B}

```

```
for opt.epoch in range(0, 199, 3):
  os.chdir('/home/huangx/disk4T/zhangy/G2/GAN2/implementations/
   pix2pix/')
  generator.load_state_dict(torch.load('saved_models/ours_500/
   generator_%d.pth' % opt.epoch))
  os.chdir('/home/huangx/disk4T/zhangy/G2/GAN2/data/ours_500/test')
  files = glob.glob('*.png')
  for file in files:
    print(file)
    imgs = get_test_img(file, transform_image)
    real_A = Variable(imgs['img_B'])
    real_B = Variable(imgs['img_A'])
    fake_B = generator(real_A)
    img_sample = torch.cat((real_B.data, fake_B.data), -1)
    os.chdir('/home/huangx/disk4T/zhangy/G2/GAN2/implementations/
   pix2pix/')
    os.makedirs('test/ours_500/%s' % opt.epoch, exist_ok=True)
    save_image(img_sample, 'test/ours_500/%d/%s' % (opt.epoch, file)
   , normalize=True)
    print('%s is tested' % file)
```

Nov 25 , 2019