RealSR算法

```
"""
CutBlur
Copyright 2020-present NAVER corp.
MIT license
"""
import sys
sys.path.append("../")
import importlib
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import torch
import torch.nn.functional as F
import model.edsr as edsr
%matplotlib inline

# options for EDSR
# 初始化大小
class Opt:
    scale = 4
    num_blocks = 32
    num_channels = 256
    res_scale = 0.1

#返回一个张量值
def im2tensor(im):
    np_t = np.ascontiguousarray(im.transpose((2, 0, 1)))    #外层函数返回一个可以指定的dtype类型，内层函数将原矩阵按
#201列顺序排列
    tensor = torch.from_numpy(np_t).float()                 #将np_t举证转换成张量，对张量赋值也是对np_t赋值
    return tensor

def tensor2im(tensor):
    tensor = tensor.detach().squeeze(0)                     #将张量中的零维度去掉
    im = tensor.clamp(0, 255).round().cpu().byte().permute(1, 2, 0).numpy()#GPU张量转换成cpu张量
    return im

opt = Opt()
dev = torch.device("cuda" if torch.cuda.is_available() else "cpu") #利用gpu计算，如果无法使用默认的2号gpu则使用cpu
#计算
#加载模型
net_base = edsr.Net(opt).to(dev)
net_moa = edsr.Net(opt).to(dev)

#加载数据
path_image = "./inputs/Nikon_006_HR.png"
path_base = "<directory_of_pt>/RealSR_EDSR_X4_base.pt"
path_moa = "<directory_of_pt>/RealSR_EDSR_X4_moa.pt"
```

```
#数据预处理，数据存储位置读取，加载计算模型
state_base = torch.load(path_base, map_location=lambda storage, loc: storage)
state_moa = torch.load(path_moa, map_location=lambda storage, loc: storage)
net_base.load_state_dict(state_base)
net_moa.load_state_dict(state_moa)

LR = io.imread(path_image)                          #利用opencv通过路径读取图像
LR_tensor = im2tensor(LR).unsqueeze(0).to(dev)  #图像转换成张良

with torch.no_grad():
    #LR图片转换成一维张量
    SR_base = tensor2im(net_base(LR_tensor))
    SR_moa = tensor2im(net_moa(LR_tensor))

#图像显示模板大小
LR_plot = LR[100:400, 750:1100] / 255
SR_base_plot = SR_base[100:400, 750:1100] / 255
SR_moa_plot = SR_moa[100:400, 750:1100] / 255

#利用算法进行图像计算，算法已经写入复现文档中
diff_SR_base = (LR_plot-SR_base_plot).mean(2) * 10
diff_SR_moa = (LR_plot-SR_moa_plot).mean(2) * 10

#对已迭代好的图像进行处理，输出在屏幕上
f, axarr = plt.subplots(3, 2, figsize=(10, 12))
axarr[0, 0].imshow(LR_plot)
axarr[0, 0].set_title("Input (HR)", fontsize=18)
axarr[0, 0].axis("off")

axarr[0, 1].axis("off")

axarr[1, 0].imshow(SR_base_plot)
axarr[1, 0].set_title("EDSR w/o MoA", fontsize=18)
axarr[1, 0].axis("off")

axarr[1, 1].imshow(diff_SR_base, vmin=0, vmax=1, cmap="viridis")
axarr[1, 1].set_title("EDSR w/o MoA (Δ)", fontsize=18)
axarr[1, 1].axis("off")

axarr[2, 0].imshow(SR_moa_plot)
axarr[2, 0].set_title("EDSR w/ MoA", fontsize=18)
axarr[2, 0].axis("off")

axarr[2, 1].imshow(diff_SR_moa, vmin=0, vmax=1, cmap="viridis")
axarr[2, 1].set_title("EDSR w/ MoA (Δ)", fontsize=18)
axarr[2, 1].axis("off")

plt.show()
```

DIV2K算法

同第一种realsr算法一样的解释，这里不再重复

```
path_image = "./inputs/0869.png"
path_base = "<directory_of_pt>/DIV2K_EDSR_X4_base.pt"
path_moa = "<directory_of_pt>/DIV2K_EDSR_X4_moa.pt"
```

```
state_base = torch.load(path_base, map_location=lambda storage, loc: storage)
state_moa = torch.load(path_moa, map_location=lambda storage, loc: storage)
net_base.load_state_dict(state_base)
net_moa.load_state_dict(state_moa)

LR = io.imread(path_image)
LR_tensor = im2tensor(LR).unsqueeze(0).to(dev)

with torch.no_grad():
    SR_base = tensor2im(net_base(LR_tensor))
    SR_moa = tensor2im(net_moa(LR_tensor))


LR_plot = LR[600:900, 550:950] / 255
SR_base_plot = SR_base[600:900, 550:950] / 255
SR_moa_plot = SR_moa[600:900, 550:950] / 255

diff_SR_base = (LR_plot-SR_base_plot).mean(2) * 10
diff_SR_moa = (LR_plot-SR_moa_plot).mean(2) * 10

f, axarr = plt.subplots(3, 2, figsize=(10, 12))
axarr[0, 0].imshow(LR_plot)
axarr[0, 0].set_title("Input (HR)", fontsize=18)
axarr[0, 0].axis("off")

axarr[0, 1].axis("off")

axarr[1, 0].imshow(SR_base_plot)
axarr[1, 0].set_title("EDSR w/o MoA", fontsize=18)
axarr[1, 0].axis("off")

axarr[1, 1].imshow(diff_SR_base, vmin=0, vmax=1, cmap="viridis")
axarr[1, 1].set_title("EDSR w/o MoA (Δ)", fontsize=18)
axarr[1, 1].axis("off")

axarr[2, 0].imshow(SR_moa_plot)
axarr[2, 0].set_title("EDSR w/ MoA", fontsize=18)
axarr[2, 0].axis("off")

axarr[2, 1].imshow(diff_SR_moa, vmin=0, vmax=1, cmap="viridis")
axarr[2, 1].set_title("EDSR w/ MoA (Δ)", fontsize=18)
axarr[2, 1].axis("off")

plt.show()
```

HR\LR同时处理——CutBlur

这里和上述两种方式略有不同，需要同时对LR和HR进行处理，因此在处理张量时需要预留一定的空间，为图像恢复做准备

```
path_image_HR = "./inputs/Canon_003_HR.png"
path_image_LR = "./inputs/Canon_003_LR4.png"
path_base = "<directory_of_pt>/RealSR_EDSR_X4_base.pt"
path_moa = "<directory_of_pt>/RealSR_EDSR_X4_moa.pt"
```

```
state_base = torch.load(path_base, map_location=lambda storage, loc: storage)
state_moa = torch.load(path_moa, map_location=lambda storage, loc: storage)
net_base.load_state_dict(state_base)
net_moa.load_state_dict(state_moa)

HR = io.imread(path_image_HR)
HR_tensor = im2tensor(HR).unsqueeze(0).to(dev)

LR = io.imread(path_image_LR)
LR_tensor = im2tensor(LR).unsqueeze(0).to(dev)

# apply CutBlur
LR_tensor[..., 900:1250, :600] = HR_tensor[..., 900:1250, :600]

with torch.no_grad():
    SR_base = tensor2im(net_base(LR_tensor))
    SR_moa = tensor2im(net_moa(LR_tensor))

LR_plot = tensor2im(LR_tensor)[900:1500, 100:650] / 255
HR_plot = HR[900:1500, 100:650] / 255
SR_base_plot = SR_base[900:1500, 100:650] / 255
SR_moa_plot = SR_moa[900:1500, 100:650] / 255

diff_SR_base = (HR_plot-SR_base_plot).mean(2) * 10
diff_SR_moa = (HR_plot-SR_moa_plot).mean(2) * 10

f, axarr = plt.subplots(3, 2, figsize=(10, 14))
axarr[0, 0].imshow(LR_plot)
axarr[0, 0].set_title("Input (Cutblurred LR)", fontsize=18)
axarr[0, 0].axis("off")

axarr[0, 1].axis("off")

axarr[1, 0].imshow(SR_base_plot)
axarr[1, 0].set_title("EDSR w/o MoA", fontsize=18)
axarr[1, 0].axis("off")

axarr[1, 1].imshow(diff_SR_base, vmin=0, vmax=1, cmap="viridis")
axarr[1, 1].set_title("EDSR w/o MoA (Δ)", fontsize=18)
axarr[1, 1].axis("off")

axarr[2, 0].imshow(SR_moa_plot)
axarr[2, 0].set_title("EDSR w/ MoA", fontsize=18)
axarr[2, 0].axis("off")

axarr[2, 1].imshow(diff_SR_moa, vmin=0, vmax=1, cmap="viridis")
axarr[2, 1].set_title("EDSR w/ MoA (Δ)", fontsize=18)
axarr[2, 1].axis("off")

plt.show()
```