该文件内包含以下class和方法

```
1. class RandomVerticalFlip(object):
    """
    随机垂直翻转
    """
        def __call__(self, img):
            """
            input: img
            output: 翻转后的img
            """
2. class DeNormalize(object):
    """
    规范化
    """
        def __init__(self, mean, std):
        def __call__(self, tensor):
        """
            input: 一个tensor
            output: 规范化处理之后的tensor
            """
3. class MaskToTensor(object):
        def __call__(self, img):
        """
        input: img
        output: tensor
        """


4. class FreeScale(object):
    """
    自由缩放
    """
        def __init__(self, size, interpolation=Image.BILINEAR):
         def __call__(self, img):
            """
            input img
            output img
            """


5. class FlipChannels(object):
    """
    转换channel
    Image.open()都进来RGB图像，相当于把RGB三个通道对应的值给到BGR上，就是说，R和B通道上的值互换一下。例：berlin1_image1中红色的车会变为蓝色
    """
        def __call__(self, img):
            """
            input img
            output img
            """


 6. class RandomGaussianBlur(object):
    """
```

```
    随机高斯模糊
    """
        def __call__(self, img):
            """
            input: img
            output: 模糊后的img
            """

7.  class Compose(object):
    """
    对transforms传进来的处理依次执行
    """
        def __init__(self, transforms):
        def __call__(self, img, mask):
            """
            input: img, mask
            output: 处理后得到的img, mask
            """

8.  class RandomCrop(object):
    """
    随机裁剪
    """
        def __init__(self, size, padding=0):
        def __call__(self, img, mask):
            """
            如果传进来的图片比需要的图片大小要小的话就填充
            大的话就裁剪
            input: img, mask
            output: 裁剪后的img和mask
            """

9.  class CenterCrop(object):
    """
    裁剪出最中心部分的图片
    """
        def __init__(self, size):
        def __call__(self, img, mask):
            """
            如果传进来的图片比需要的图片要大，减成相同大小的图片
            如果传进来的图片比需要的要小，在周围padding，测试结果是补黑色
            input: img, mask
            output: 裁剪或者填充之后的img, mask
            """

10. class RandomHorizontallyFlip(object):
    """
    随机水平翻转
    这里就有一个问题了
    为什么水平翻转是要把mask一起翻转了，但是竖直反转就不要呢？
    """
        def __call__(self, img, mask):

11. class Scale(object):
    """
```

```
    只要h和w中有一个和self.size相等，直接返回
    如果h和w与self.size都不相等，把较大值变为self.,较小值按原图像等比例设置
    """
        def __init__(self, size):
        def __call__(self, img, mask):
            """
            input: img, mask
            """

12. class RandomSizedCrop(object):
    """
    随机大小裁剪
    """
        def __init__(self, size):
        def __call__(self, img, mask):
            """
            input: img, mask
            """

13. class RandomRotate(object):
    """
    随机旋转
    """
        def __init__(self, degree):
        def __call__(self, img, mask):
            """
            input: img, mask
            """

14. class RandomSized(object):
    """
    先随机resize img，  再调用Scale，和Randomcop
    """
        def __init__(self, size):
        def __call__(self, img, mask):
            """
            input: img, mask
            """

15. class SlidingCropOld(object):
    """
    对比crop_size大的图片取图像内crop_size大小的小图，并输出包含所有小图一个list
    对比crop_size小的图片，通过填充，输出crop_size大小的图片
    """
        def __init__(self, crop_size, stride_rate, ignore_label):
        def _pad(self, img, mask):
        def __call__(self, img, mask):
16. class SlidingCrop(object):
    """
    比old多输出一个小图的坐标信息的list
    """



import random
```

```python
import numpy as np
from PIL import Image, ImageOps, ImageFilter
from skimage.filters import gaussian
import torch
import math
import numbers


class RandomVerticalFlip(object):
    """
    随机垂直翻转
    """
    def __call__(self, img):
        if random.random() < 0.5:
            # 上下镜像 + 图像转置
            return img.transpose(Image.FLIP_TOP_BOTTOM)
        return img


class DeNormalize(object):
    """
    去规范化
    见到的时候记得看一下从哪读的，用在哪里的
    """
    def __init__(self, mean, std):
        self.mean = mean
        self.std = std

    def __call__(self, tensor):
        for t, m, s in zip(tensor, self.mean, self.std):
            # mul_是in-place处理
            t.mul_(s).add_(m)
        return tensor


class MaskToTensor(object):
    def __call__(self, img):
        # 就是torch.from_numpy()方法把数组转换成张量，且二者共享内存，
        # 对张量进行修改比如重新赋值，那么原始数组也会相应发生改变。
        return torch.from_numpy(np.array(img, dtype=np.int32)).long()


class FreeScale(object):
    # 双线性插值
    def __init__(self, size, interpolation=Image.BILINEAR):
        # reversed 返回一个反转的迭代器 只是一个反转h，w。
        # opencv储存图片格式是（H，W，C），而torch储存的格式是（C，H，W）
        self.size = tuple(reversed(size))  # size: (h, w)
        self.interpolation = interpolation

    def __call__(self, img):
        # img.resize需要的输入是w， h
        # 这里的img输入得是用Image.open打开的不能是opencv打开的
        # 很明显这里用的是Image.resize 不是cv2.resize 所以对输入的img的要求
        # 是使用Imag.open打开的图像
```

```
        """
        Image.open()和ci2.imread()都是用来读取的图像，但在使用过程中存在一些差别。
        cv2.imread()读取的是图像的真实数据。Image.open()函数只是保持了图像被读取的状态，
        但是图像的真实数据并未被读取，因此如果对需要操作图像每个元素，如输出某个像素的RGB值等，
        需要执行对象的load()方法读取数据。例如print(Image.open())得到的是一个状态(对象？)
        <PIL.PngImagePlugin.PngImageFile image mode=RGB size=2611x2453
        at 0x1DC9E9D1C10>
        Image.open()得到的img数据类型呢是Image对象，不是普通的数组。
        cv2.imread()得到的img数据类型是np.array()类型。
        """
        return img.resize(self.size, self.interpolation)


class FlipChannels(object):
    """
    转换channel
    Image.open()都进来RGB图像，相当于把RGB三个通道对应的值给到BGR上，就是说，R和B通道上的值
    互换一下。例：berlin1_image1中红色的车会变为蓝色
    """
    def __call__(self, img):
        # a[::-1]相当于 a[-1:-len(a)-1:-1]，也就是从最后一个元素到第一个元素复制一遍。
        img = np.array(img)[:, :, ::-1]
        # Image.fromarray就是实现array到image的转换
        return Image.fromarray(img.astype(np.uint8))


class RandomGaussianBlur(object):
    """
    随机高斯模糊
    """
    def __call__(self, img):
        # 为什么要用0.15啊 sigma默认为1
        sigma = 0.15 + random.random() * 1.15
        # gaussian是用的skimage-filter里面的gaussian
        # multichannel: bool，可选（默认值：无）图像的最后一个轴是否被解释为多个通道。
        # 如果属实，每个通道都单独过滤（通道不混合在一起）
        # https://cloud.tencent.com/developer/section/1414823
        # 警告！！！ 这个用法将要在version1.0中被更改！项目内需要注意！
        # FutureWarning: `multichannel` is a deprecated argument name for
        # `gaussian`. It will be removed in version 1.0. Please use
        # `channel_axis` instead.
        # 建议把multichannel改为channel_axis=True 经测试效果一致
        blurred_img = gaussian(np.array(img), sigma=sigma, multichannel=True)
        # 为什么要乘 255啊：在上一步经过高斯模糊处理之后blurred_img里面的值会变成这种
        #   [0.10980392 0.10980392 0.1254902 ]
        #   [0.25098039 0.26666667 0.26666667]
        #   [0.36078431 0.39215686 0.39215686]
        # gaussian(np.array(img), sigma=sigma, multichannel=True)
        blurred_img *= 255
        return Image.fromarray(blurred_img.astype(np.uint8))


class Compose(object):
    """
    这个transforms是从哪进来的啊，还未见到
```

```
    """
    def __init__(self, transforms):
        self.transforms = transforms

    def __call__(self, img, mask):
        assert img.size == mask.size
        # 对img， mask依次做transforms内的预处理
        for t in self.transforms:
            img, mask = t(img, mask)
        return img, mask


class RandomCrop(object):
    """
    随机裁剪
    """
    def __init__(self, size, padding=0):
        # 如果size是一个数，转为int，int的元组
        if isinstance(size, numbers.Number):
            self.size = (int(size), int(size))
        else:
            self.size = size
        self.padding = padding

    def __call__(self, img, mask):
        if self.padding > 0:
            # 加边框 self.padding是以像素为单位的边框宽度
            img = ImageOps.expand(img, border=self.padding, fill=0)
            mask = ImageOps.expand(mask, border=self.padding, fill=0)

        assert img.size == mask.size
        w, h = img.size
        th, tw = self.size
        if w == tw and h == th:
            return img, mask
        # 如果传进来的图片比需要的图片大小要小的话就填充
        if w < tw or h < th:
            return img.resize((tw, th), Image.BILINEAR), mask.resize((tw, th),

Image.NEAREST)
        # 如果比传进来的图片比需要的图片要大，随机裁减成相同大小的图片
        x1 = random.randint(0, w - tw)
        y1 = random.randint(0, h - th)
        return img.crop((x1, y1, x1 + tw, y1 + th)), mask.crop(
            (x1, y1, x1 + tw, y1 + th))


class CenterCrop(object):
    """
    裁剪出最中心部分的图片
    如果传进来的图片比需要的图片要大，减成相同大小的图片
    如果传进来的图片比需要的要小，在周围padding，测试结果是补黑色
    """
    def __init__(self, size):
        if isinstance(size, numbers.Number):
```

```
            self.size = (int(size), int(size))
        else:
            self.size = size

    def __call__(self, img, mask):
        assert img.size == mask.size
        w, h = img.size
        th, tw = self.size
        x1 = int(round((w - tw) / 2.))
        y1 = int(round((h - th) / 2.))
        return img.crop((x1, y1, x1 + tw, y1 + th)), mask.crop(
            (x1, y1, x1 + tw, y1 + th))


class RandomHorizontallyFlip(object):
    """
    随机水平翻转
    这里就有一个问题了
    为什么水平翻转是要把mask一起翻转了，但是竖直反转就不要呢？
    """
    def __call__(self, img, mask):
        if random.random() < 0.5:
            return img.transpose(Image.FLIP_LEFT_RIGHT), mask.transpose(
                Image.FLIP_LEFT_RIGHT)
        return img, mask


class Scale(object):
    """
    只要h和w中有一个和self.size相等，直接返回
    如果h和w与self.size都不相等，把较大值变为self.,较小值按原图像等比例设置
    """
    def __init__(self, size):
        self.size = size

    def __call__(self, img, mask):
        assert img.size == mask.size
        w, h = img.size
        # 只要h和w中有一个和self.size相等，直接返回
        if (w >= h and w == self.size) or (h >= w and h == self.size):
            return img, mask
        # 如果h和w与self.size都不相等，把较大值变为self.,较小值按原图像等比例设置
        if w > h:
            ow = self.size
            oh = int(self.size * h / w)
            return img.resize((ow, oh), Image.BILINEAR), mask.resize((ow, oh),

Image.NEAREST)
        else:
            oh = self.size
            ow = int(self.size * w / h)
            return img.resize((ow, oh), Image.BILINEAR), mask.resize((ow, oh),

Image.NEAREST)
```

```python
class RandomSizedCrop(object):
    """
    随机大小裁剪
    """
    def __init__(self, size):
        self.size = size

    def __call__(self, img, mask):
        assert img.size == mask.size
        for attempt in range(10):
            area = img.size[0] * img.size[1]
            # random.uniform(x, y) 方法将随机生成一个实数，它在 [x,y] 范围内。
            target_area = random.uniform(0.45, 1.0) * area
            # 一个随机的纵横比
            aspect_ratio = random.uniform(0.5, 2)
            # S=wxh
            # aspect_ratio = w/ h
            # S x aspect_ration = w x h x w/h = w^2
            # 所以这里求的是按照随机比例缩小之后，按照随机纵横比得到的w和h
            w = int(round(math.sqrt(target_area * aspect_ratio)))
            h = int(round(math.sqrt(target_area / aspect_ratio)))

            if random.random() < 0.5:
                w, h = h, w
            # 如果新得到的w和h都比原图的w和h小随机裁剪得到小图
            if w <= img.size[0] and h <= img.size[1]:
                x1 = random.randint(0, img.size[0] - w)
                y1 = random.randint(0, img.size[1] - h)

                img = img.crop((x1, y1, x1 + w, y1 + h))
                mask = mask.crop((x1, y1, x1 + w, y1 + h))
                assert (img.size == (w, h))

                # 再把小图resize到目标大小
                return img.resize((self.size, self.size),
                                  Image.BILINEAR), mask.resize(
                    (self.size, self.size),
                    Image.NEAREST)

        # Fallback
        scale = Scale(self.size)
        crop = CenterCrop(self.size)
        return crop(*scale(img, mask))


class RandomRotate(object):
    def __init__(self, degree):
        self.degree = degree

    def __call__(self, img, mask):
        rotate_degree = random.random() * 2 * self.degree - self.degree
        return img.rotate(rotate_degree, Image.BILINEAR), mask.rotate(
            rotate_degree, Image.NEAREST)
```

```
class RandomSized(object):
    def __init__(self, size):
        self.size = size
        self.scale = Scale(self.size)
        self.crop = RandomCrop(self.size)

    def __call__(self, img, mask):
        assert img.size == mask.size

        w = int(random.uniform(0.5, 2) * img.size[0])
        h = int(random.uniform(0.5, 2) * img.size[1])

        img, mask = img.resize((w, h), Image.BILINEAR), mask.resize((w, h),

 Image.NEAREST)

        return self.crop(*self.scale(img, mask))


class SlidingCropOld(object):
    """
    对比crop_size大的图片取图像内crop_size大小的小图，并输出包含所有小图一个list
    对比crop_size小的图片，通过填充，输出crop_size大小的图片
    """
    def __init__(self, crop_size, stride_rate, ignore_label):
        self.crop_size = crop_size
        self.stride_rate = stride_rate
        self.ignore_label = ignore_label

    def _pad(self, img, mask):
        """
        填充图片
        """
        h, w = img.shape[: 2]
        # 判断crop_size是不是要比图像原本的h和w要大。
        pad_h = max(self.crop_size - h, 0)
        pad_w = max(self.crop_size - w, 0)
        # 使用np.pd 进行数据填充（）里四个值顺序为上下左右
        img = np.pad(img, ((0, pad_h), (0, pad_w), (0, 0)), 'constant')
        mask = np.pad(mask, ((0, pad_h), (0, pad_w)), 'constant',
                      constant_values=self.ignore_label)
        return img, mask

    def __call__(self, img, mask):
        """
        对比crop_size大的图片取图像内crop_size大小的小图，并输出包含所有小图一个list
        对比crop_size小的图片，通过填充，输出crop_size大小的图片
        """
        assert img.size == mask.size

        w, h = img.size
        long_size = max(h, w)

        img = np.array(img)
```

```
        mask = np.array(mask)

        # 如果输入图像h，w中较大的那个大于crop_size
        if long_size > self.crop_size:
            # math.ceil 向上取整
            stride = int(math.ceil(self.crop_size * self.stride_rate))
            # 几步跨完h和w
            h_step_num = int(
                math.ceil((h - self.crop_size) / float(stride))) + 1
            w_step_num = int(
                math.ceil((w - self.crop_size) / float(stride))) + 1
            img_sublist, mask_sublist = [], []
            for yy in range(h_step_num):
                for xx in range(w_step_num):
                    sy, sx = yy * stride, xx * stride
                    ey, ex = sy + self.crop_size, sx + self.crop_size
                    # 图片内的crop_size大小的小图
                    img_sub = img[sy: ey, sx: ex, :]
                    mask_sub = mask[sy: ey, sx: ex]
                    # 对大小不足crop_size的进行填充
                    img_sub, mask_sub = self._pad(img_sub, mask_sub)
                    img_sublist.append(
                        Image.fromarray(img_sub.astype(np.uint8)).convert(
                            'RGB'))
                    mask_sublist.append(
                        Image.fromarray(mask_sub.astype(np.uint8)).convert('P'))
            return img_sublist, mask_sublist
        else:
            img, mask = self._pad(img, mask)
            img = Image.fromarray(img.astype(np.uint8)).convert('RGB')
            mask = Image.fromarray(mask.astype(np.uint8)).convert('P')
            return img, mask


class SlidingCrop(object):
    """
    比old多输出一个小图的坐标信息的list
    """
    def __init__(self, crop_size, stride_rate, ignore_label):
        self.crop_size = crop_size
        self.stride_rate = stride_rate
        self.ignore_label = ignore_label

    def _pad(self, img, mask):
        h, w = img.shape[: 2]
        pad_h = max(self.crop_size - h, 0)
        pad_w = max(self.crop_size - w, 0)
        img = np.pad(img, ((0, pad_h), (0, pad_w), (0, 0)), 'constant')
        mask = np.pad(mask, ((0, pad_h), (0, pad_w)), 'constant',
                      constant_values=self.ignore_label)
        return img, mask, h, w

    def __call__(self, img, mask):
        assert img.size == mask.size
```

```
        w, h = img.size
        long_size = max(h, w)

        img = np.array(img)
        mask = np.array(mask)

        if long_size > self.crop_size:
            stride = int(math.ceil(self.crop_size * self.stride_rate))
            h_step_num = int(
                math.ceil((h - self.crop_size) / float(stride))) + 1
            w_step_num = int(
                math.ceil((w - self.crop_size) / float(stride))) + 1
            img_slices, mask_slices, slices_info = [], [], []
            for yy in range(h_step_num):
                for xx in range(w_step_num):
                    sy, sx = yy * stride, xx * stride
                    ey, ex = sy + self.crop_size, sx + self.crop_size
                    img_sub = img[sy: ey, sx: ex, :]
                    mask_sub = mask[sy: ey, sx: ex]
                    img_sub, mask_sub, sub_h, sub_w = self._pad(img_sub,
                                                                mask_sub)
                    img_slices.append(
                        Image.fromarray(img_sub.astype(np.uint8)).convert(
                            'RGB'))
                    mask_slices.append(
                        Image.fromarray(mask_sub.astype(np.uint8)).convert('P'))
                    slices_info.append([sy, ey, sx, ex, sub_h, sub_w])
            return img_slices, mask_slices, slices_info
        else:
            img, mask, sub_h, sub_w = self._pad(img, mask)
            img = Image.fromarray(img.astype(np.uint8)).convert('RGB')
            mask = Image.fromarray(mask.astype(np.uint8)).convert('P')
            return [img], [mask], [[0, sub_h, 0, sub_w, sub_h, sub_w]]
```