代码环境

- 安装python3
- 安装tensorflow 1.x（注意不要最新2.0版本）
- 安装tensorflow工具包 Neurogym

训练前的准备

1. 数据分包
    - 代码里包含我写好的脚本 gen_flist.py

      运行

      ```
      python gen_flist.py --folder_path folderpath --train_filename
      train_filename --validation_filename validation_filename
      ```

    - 这个脚本能够自动切分数据集成训练集和验证集。使用前请修改default地址。

      ```
      parser.add_argument('--
      folder_path',default='default',type=str,help='The folder path')
      parser.add_argument("--
      train_filename",default='default',type=str,help='The train filename')
      parser.add_argument('--validation_filename', default='default',
      type=str,help='The validation filename.')
      ```

2. mask图片以及masked图片生成
    - 参考我代码里的 imagecreate.py 脚本。

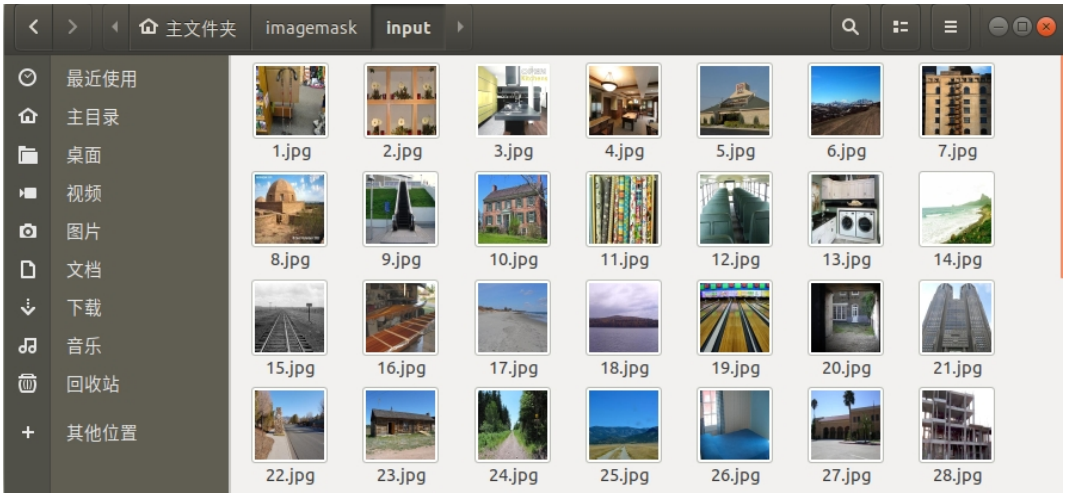
      以生成矩形mask块为例，运行

      ```
      python imagecreate.py --input_dirimg input_dirimg --output_dirmask
      output_dirmask --output_dirmasked output_dirmasked --HEIGHT X --WIDTH Y
      ```

      该代码将 input_dirimg 地址下图片随机生成矩形mask块，高度为X宽度为Y，并将被遮挡的masked图片保存在 output_dirmasked 地址下，将生成的mask图片保存在 output_dirmask 地址下。

      代码效果如下：

      输入图片

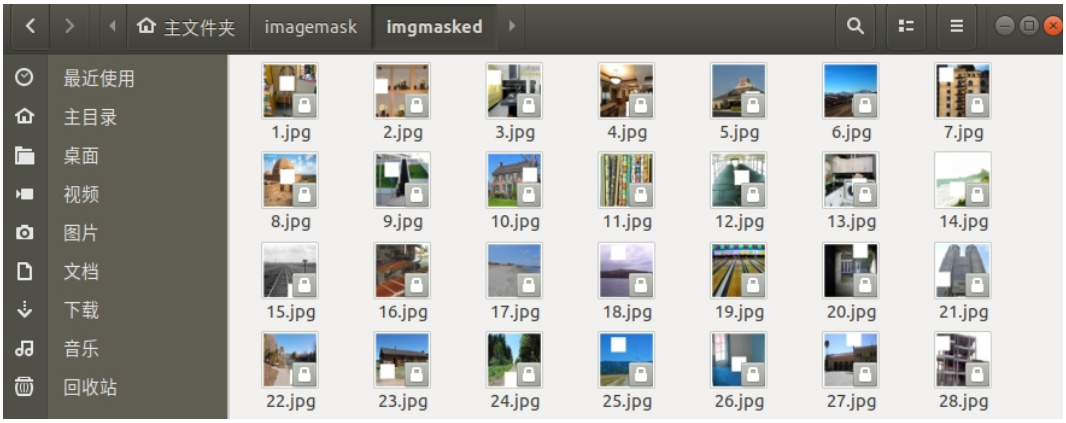


运行

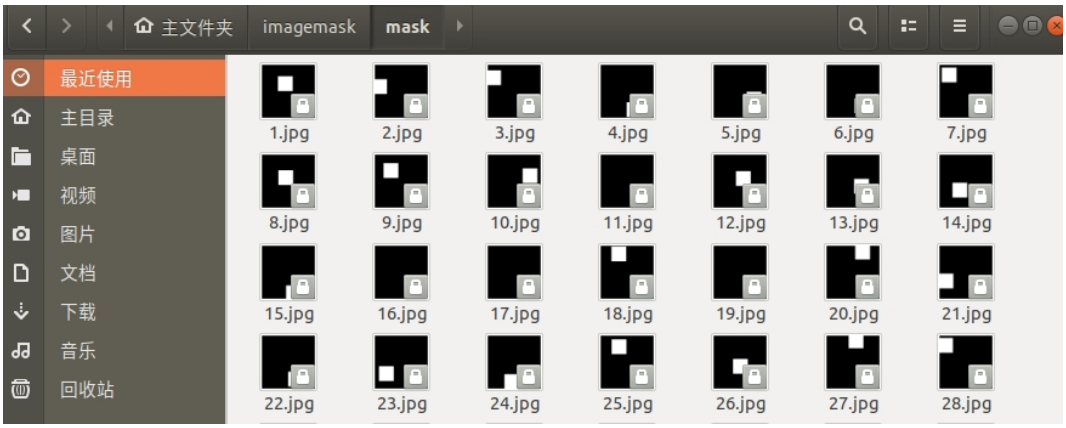
```
imagecreate.py --input_dirimg /home/baoge/imagemask/input --
output_dirmask /home/baoge/imagemask/mask --output_dirmasked
/home/baoge/imagemask/imgmasked --HEIGHT 64 --WIDTH 64
```

输出文件夹：

生成的随机masked图片。



生成随机的mask，对应相应的位置。



- imagecreate.py 脚本包含其他很多我已经写好的mask生成函数。

  包括批量生成mask图片，单张生成，随即不规则掩膜（随机线，随机椭圆形，随机圆形等），使用时请修改最下方main函数或者调用特定函数。

```
if __name__ == '__main__':
    config = parser.parse_args()
    get_path(config)
    # 单张图像生成mask
    # img = './data/test.jpg'
    # masked_img,mask = load_mask(img,config,True)
    img2maskedImg(config.input_dirimg)

    # 矩形特殊处理 处理同样shape的图片(256,256,3)
    # img = './examples/celeba/000042.jpg'
    #img = config.img
    #masked_img, mask = random_mask_rect(img, config)
```

训练

- 修改 inpaint.yml，修改里面的默认数据，包括dataflist存放地址等。
- 训练模型

运行

```
python train.py
```

如果需要继续训练，请修改inpaint.yml里的MODEL_RESTORE存放地址。

并运行

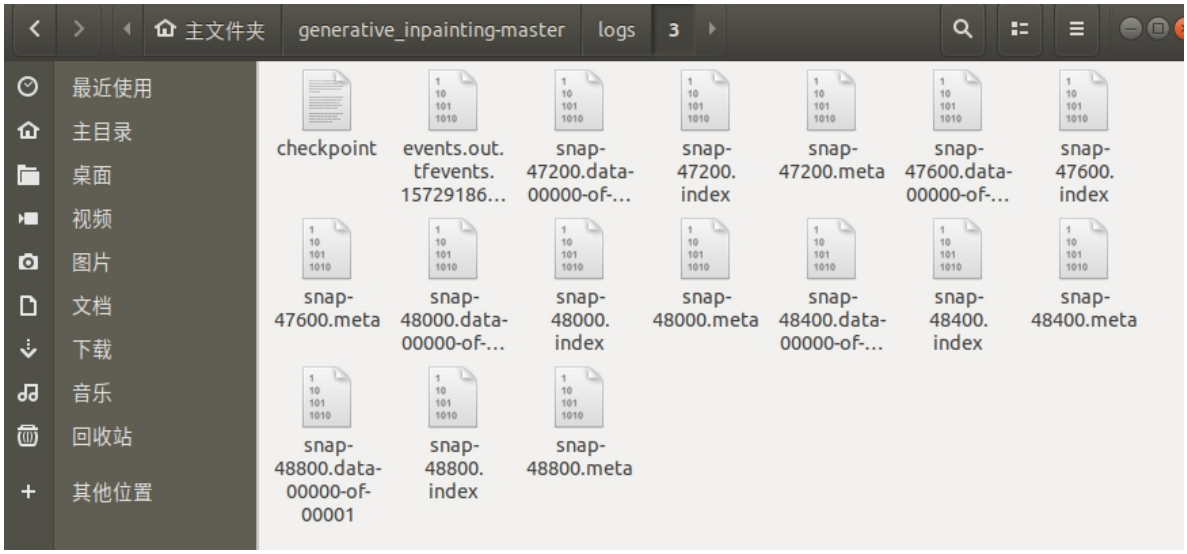
```
python train.py
```

- 测试

运行

```
python test.py --image maskedimage --mask maskimage --output outputimage --
checkpoint model_logs/your_model_dir
```

输出模型（我把自己的模型放在了 logs/3/ 文件夹下）：



训练后的数据收集

- 这里我主要准备了两个脚本可以运行

1. metrics.py

- 全局比较图片的修复效果，将修复后的图片与原图进行比较。

运行

```
python metrics.py --data-path inputimage --output-path outputimage
```

运行效果如下：

```
(project) root@baoge-MS-7B17:~/generative_inpainting-master# python metrics.py -
-data-path /home/baoge/imagemask/input --output-path /home/baoge/imagemask/outpu
t1
[data_path] = /home/baoge/imagemask/input
[output_path] = /home/baoge/imagemask/output1
[debug] = 0
PSNR: 29.9905 PSNR Variance: 22.1026 SSIM: 0.9589 SSIM Variance: 0.0004 MAE: 0.0
081 MAE Variance: 0.0000
```

2. metrics_part.py

- 只比较图片的修复区域的效果。只取修复区域的图像进行比较。

运行

```
python metrics_part.py --data-path inputimage --output-path outputimage --mask-
path maskimage
```

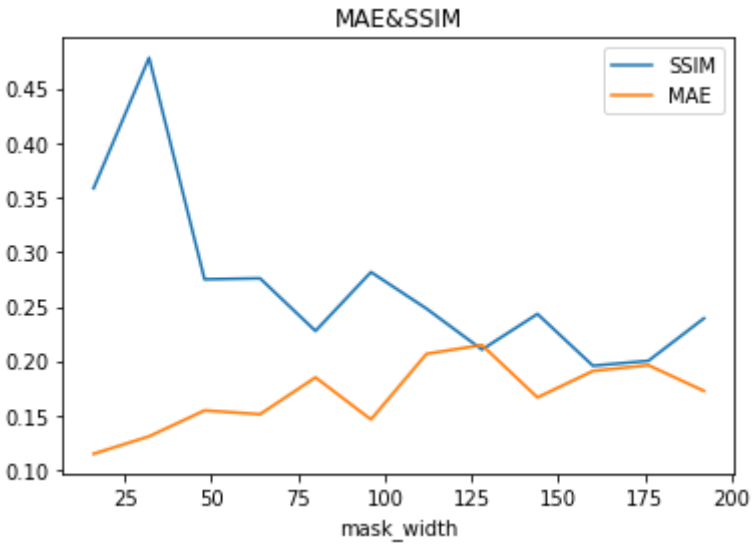
运行效果如下：

```
(project) root@baoge-MS-7B17:~/generative_inpainting-master# python metrics_part
.py --data-path /home/baoge/imagemask/input --output-path /home/baoge/imagemask/
output1 --mask-path /home/baoge/imagemask/mask
[data_path] = /home/baoge/imagemask/input
[mask_path] = /home/baoge/imagemask/mask
[output_path] = /home/baoge/imagemask/output1
[debug] = 0
PSNR: 19.0412 PSNR Variance: 30.4586 SSIM: 0.4405 SSIM Variance: 0.0542 MAE: 0.1
145 MAE Variance: 0.0046
```

模型结果

- 我将之前留存的具体数据保存在datavalue.py 文件中作为参考。
- 模型修复图像的效果如下：

- 修复图像的量化比较：



参考文献

1. Generative Image Inpainting with Contextual Attention https://arxiv.org/abs/1801.07892