图形学实验: 生成中间帧

给定初始图片和结束图片，生成中间的N帧，使得首尾自然过渡

Table of Contents

开发环境

- **开发环境：** macOS Mojave 10.14.6
- **开发软件：** PyCharm 2019.1.3
- **开发语言：** python

如何运行

- 将项目文件夹拷贝到本地环境
- 运行`src/GenerateIntermediateFrames.py`
- 在`Resources/Result/`目录下可查看到生成的中间帧图片
  - `framexx.png`: 为中间帧图片，其中xx为该帧编号
  - `result.gif`: 为该文件夹中所有帧图片生成的gif图片(方便观察处理结果)
- ***可使用自定义图片替换`Resources/Origin/`目录下的原始图片；并且请修改`src/GenerateIntermediateFrames.py` -> `main()`函数中`path_cat`和`path_tiger`的路径***

实验结果

测试用例一

原图

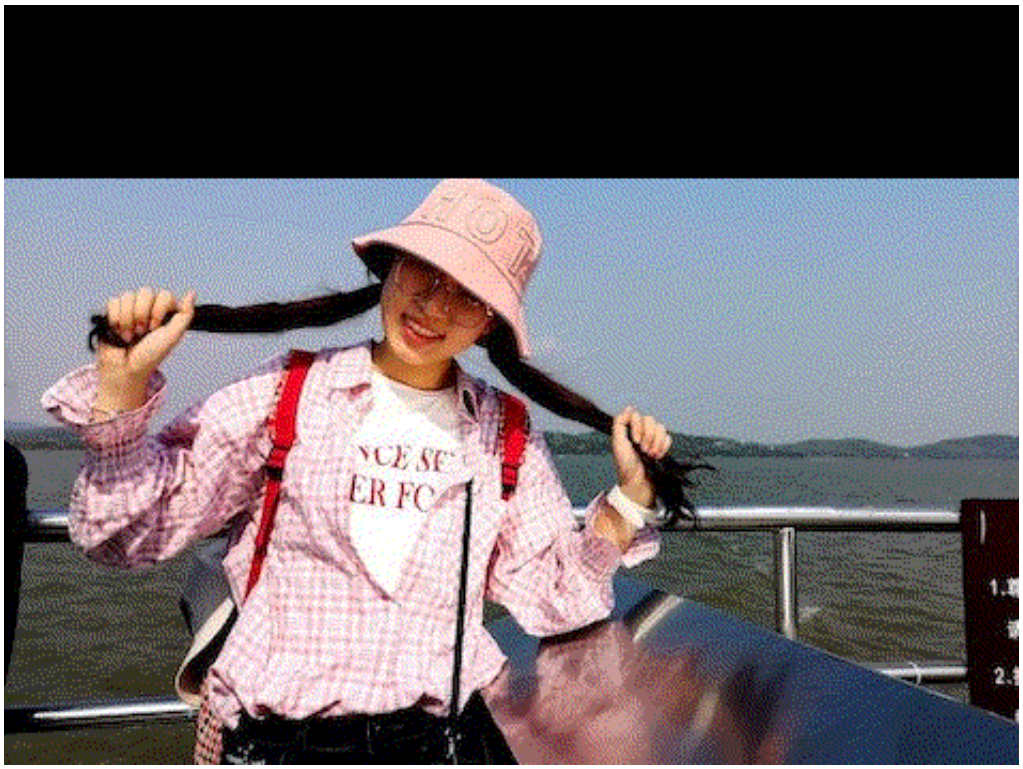
生成的中间帧(转化为gif)

测试用例二

考虑到机器性能和生成动图大小，图片有压缩

原图

生成的中间帧(转化为gif)



具体实现

读取图片

使用`cv2.imread()`将图片读取成矩阵

```
'''读取原始图片'''
def readOriginImg(path_cat, path_tiger):
  I_cat = cv2.imread(path_cat)
  I_tiger = cv2.imread(path_tiger)

  return I_cat, I_tiger
```

```
# 读取的图片数据结构
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  ...
  [ 84 103 112]
  [103 122 128]
  [167 178 181]]]
```

生成中间帧(线性插值算法)

- 取两图片最大的长宽作为目标图片的长宽

```
# 取两图片最大的长宽作为目标图片的长宽
width, height = max(I_cat.shape[0], I_tiger.shape[0]), max(I_cat.shape[1],
I_tiger.shape[1])
```

直接使用三维数组

- 构建 (N+2) * width * height 三纬结果矩阵

```
result_frames = [[[0 for _ in range(width)] for _ in range(height)] for _
in range(N+2)]
```

- 使用三重循环进行中间帧生成

```
for k in range(0, N+2):          # 帧循环

  t = k/(1+N)

  # 对二维图片矩阵遍历
  for x in range(width):
    for y in range(height):
      result_frames[k][x][y] = (1-t)*I_cat[x][y] + t*I_tiger[x][y]    # 线
性插值公式
```

使用numpy数组

将矩阵写出成图片

```
'''将矩阵写出成图片'''
def writeResultFrames(result_frames, dirname, multi_thread=False):
  for index, frame in enumerate(result_frames):
    filename = 'frame' + str(index) + '.png'
```

直接写出

```
else:
  cv2.imwrite(dirname + filename, np.float32(frame))
```

使用多线程处理复杂任务

```
'''将矩阵保存成图片 - 线程类'''
class FrameHandler(Thread):
  def __init__(self, dirname, filename, frame):
    super().__init__()
    self._dirname = dirname
    self._filename = filename
    self._frame = frame

    def run(self):
      imwrite(self._dirname+self._filename, np.float32(self._frame))
```

```
if multi_thread:
  frame_thread = FrameHandler(dirname, filename, frame)
  frame_thread.start()
  frame_thread.join()
```

将一组帧图片保存为Gif

- 对结果目录中的所有文件进行过滤(只读入生成的中间帧)
- 对读入的中间帧按照先后进行排序
- 调用`image.mimsave()`方法将一组帧图片保存为Gif

```
'''
Resources/Result/目录下有多张图片
图片格式: framexx.png
'''
import os
import imageio

def myImage2Gif(dirname):
  frames = list(filter(lambda x: x[0:5] == 'frame' and x[-4:] == '.png',
os.listdir(dirname)))  # 将不符合命名要求的图片过滤掉(MacOS)会默认创建一些文件
```

```
  frames.sort(key=lambda x: int(x[5:-4]))      # 按照图片编号进行排序

  imgs = []
  for frame in frames:
    img = Image.open(dirname + frame)
    imgs.append(img)

  imgs[0].save(dirname + 'result.gif', save_all=True, append_images=imgs,
duration=300)
```

实验分析

使用普通三维数组和numpy数组的效率差异

```
start = time()

# 生成指定数量的中间帧
result_frames = generateFrame(I_cat, I_tiger, N)

end = time()
print(end-start)
```

该测试使用的是测试用例一，生成的中间帧数量`N = 100`

仅测试`generateFrame()`函数的时间，用于反映两种算法对于处理图片矩阵，生成中间帧的效率差异：

- 使用普通三维数组
  - 结果数据结构的定义：

    ```
    result_frames = [[[0 for _ in range(width)] for _ in range(height)]
    for _ in range(N + 2)]
    ```

  - 线性插值公式的使用：

    ```
    result_frames[k][x][y] = (1 - t) * I_cat[x][y] + t * I_tiger[x][y]
    ```

  - 耗时：16.906198024749756 s
- 使用numpy数组
  - 结果数据结构的定义：

    ```
    result_frames = np.zeros((N + 2, width, height, 3))      # 对于图片矩阵中
    的每一个像素进行数值运算
    ```

  - 线性插值公式的使用：

```
result_frames[k] = (1 - t) * I_cat + t * I_tiger          # 直接对于图片矩阵进行矩阵运算
```

- 耗时：0.0948951244354248 s

从耗时中可以明显的看到使用numpy数组对于性能的提升，提升倍数达到了惊人的178倍；这提示我将来凡事观察到数值元算可以转换为直接对矩阵进行运算时，要果断采用numpy库相关函数进行科学计算，效率会得到很大的提升

使用多线程与直接写出的效率差异

```
start = time()

# 将结果矩阵写出成图片
writeResultFrames(result_frames, path_result, multi_thread=True)

end = time()
print(end - start)
```

该测试使用的是测试用例一，生成的中间帧数量`N = 1000`

仅测试`writeResultFrames()`函数的时间，通过传入参数`multi_thread`不同的值，用于反映两种算法对于将矩阵写出成图片的效率差异：

- 直接写出
  - 耗时：2.9145750999450684 s
- 使用多线程
  - 耗时：3.5594019889831543 s

按道理，采用多线程应该会提高效率，但是将数据量从100上升到1000，仍是直接写出效率更高，猜想可能是使用的原始图片像素太小，且生成的中间帧数量还是不够巨大，无法发挥出多线程的效率，反而多线程的算法中要花很多时间在函数调用和类的创建上；此外还可能跟我把线程封装成类，并且独立写在一个.py文件中让主文件调用有关。在系统调用中浪费太多时间，反而在真正能发挥多线程性能时数据量又不是很大，导致效果并不好。

遇到的问题 & 解决方案

python构建三维数组

- 初始时结果数组的定义如下

```
result_frames = [[0 for _ in range(width)] for _ in range(height)] * (N+2)
```

- 测试该数组的结构如下，觉得没有任何不妥

```
[[[0, 0], [0, 0], [0, 0]], [[0, 0], [0, 0], [0, 0]], [[0, 0], [0, 0], [0,
0]], [[0, 0], [0, 0], [0, 0]]]
```

- 但是下一步进行中间帧生成后，写出的结果矩阵却怎么也不对，生成的图片非常奇怪；仔细找了很久也没发现有任何不妥，于是读了很多篇关于python多维数组的文章，最终通过如下的测试终于找到了问题所在
- 测试代码如下，遍历三位数组的每一个元素，从小到大递增的赋值，理论上最终数组中应该是从0～23的数字

```
index = 0
for k in range(0,N+2):
    for x in range(height):
        for y in range(width):
            print(k,x,y)
            result_frames[k][x][y] = index
            index += 1
print(result_frames)
```

- 测试结果如下

```
[[[18, 19], [20, 21], [22, 23]], [[18, 19], [20, 21], [22, 23]], [[18,
19], [20, 21], [22, 23]], [[18, 19], [20, 21], [22, 23]]]
```

- 从测试中可以看到数组的第二三纬度的值每次都会同时修改，导致`[[18, 19], [20, 21], [22, 23]]`这组值反复出现，终于找到问题的所在
- 找到准确问题后在网上进一步了解，发现时python深拷贝浅拷贝的问题，在创建数组中的`*(N+2)`造成了对上一层两维数组的浅拷贝
- **修改方法：**

```
result_frames = [[[0 for _ in range(width)] for _ in range(height)] for _
in range(N + 2)]

# 或直接使用numpy数组
result_frames = np.zeros((N + 2, width, height, 3))
```

原始图片尺寸不匹配

- 在最开始的程序中，如果开始的两张图片尺寸不同，则会报错

```
ValueError: operands could not be broadcast together with shapes
(826,661,3) (640,640,3)
```

- 是由于python中广播机制的存在导致无法对尺寸不匹配的矩阵进行运算
- 所以必须在程序中对矩阵进行尺寸的重定义，是的不同尺寸的矩阵可以进行运算

- **修改方法：**
  - 取两图片最大的长宽作为目标图片的长宽

    ```
    width, height = max(I_cat.shape[0], I_tiger.shape[0]),
    max(I_cat.shape[1], I_tiger.shape[1])
    ```

  - 分别将两张图片进行尺寸的重新调整

    ```
    I_cat = np.resize(I_cat, (width, height, 3))
    I_tiger = np.resize(I_tiger, (width, height, 3))
    ```

将一组帧图片保存为Gif

- 生成的N+2张图片十分不便于观察自然过渡的效果，因此打算使用python将所有图片转换成一个gif
- 查阅了python库函数，发现大致有三种方法可以实现一组图片转gif
  - `imageio.mimsave()`
  - `images2gif`库中的`writeGif()`
  - `PIL.Image`库中的`save()`
- 分别使用三种方法试验之后发现
  - 第一种方法：效率有点低下，在合并10几张图片时无任何问题，但合并很多张图片容易卡死，造成无法生成gif图片
  - 第二种方法：该库是python2中的库，不再支持python3

    将`/Library/Frameworks/Python.framework/Versions/3.7/lib/images2gif`中的

    ```
    for im in images:
      palettes.append( getheader(im)[1] )
    ```

    替换成

    ```
    for im in images:
      palettes.append( im.palette.getdata()[1])
    ```

    仍然会出现问题
  - 第三种方法：有时候会只在首位两张图片之间闪烁
- 最终使用第三种方法，将`duration`设置为300，效果较好
- 同时由于我的架构是生成所有中间帧图片，再转换为gif，所以要读取整个结果文件夹中的文件
- 如果直接读取，顺序错乱，这样制作的动图无法观察到变化

```
['frame52.png', 'frame46.png', 'frame91.png', 'frame85.png',
'frame84.png', 'frame90.png', 'frame47.png', 'frame53.png', 'frame79.png',
..., 'frame77.png']
```

- 对文件列表进行排序，但是有一些macOS系统自带的文件，并且排序是按照字符串的字典序排序，这样也同样是无法使用的

```
[.DS_Store, 'frame0.png', 'frame1.png', 'frame10.png', 'frame100.png',...
]
```

  - 首先对数据进行清洗，将无关的文件过滤掉

```
frames = list(filter(lambda x: x[0:5] == 'frame' and x[-4:] == '.png',
os.listdir(dirname)))
```

  - 按照图片标号进行排序：自己创建排序函数的key

```
frames.sort(key=lambda x: int(x[5:-4]))
```

项目结构

```
.
├── README.md
├── Resources
|   ├── Origin
|   |   ├── cat.png
|   |   ├── her1.jpeg
|   |   ├── her2.jpeg
|   |   └── tiger.png
|   └── Result
|       ├── catResult.gif
|       └── herResult.gif
├── doc
|   └── 生成中间帧实验文档.pdf
└── src
    ├── FrameHandler.py
    ├── GenerateIntermediateFrames.py
    └── myImage2Gif.py
```

关于作者

项目	信息
姓名	张喆
学号	1754060
指导老师	贾金原老师
上课时间	周一 5、6、7节
联系方式	doubleZ0108@163.com