《计算机视觉基础》大作业报告

基于OpenCV的绘画图像线稿提取

> 作者：徐宗迪
>
> 班级：计科1405
>
> 学号：1030414518
>
> 指导老师：宋晓宁

> 日期：2017-6-5

实验背景

关于线稿的价值

对于众多美术绘画初学者来说，绘制过程中，线稿是一道不容易翻越的坎。

简单的说，线稿是反复多次打草稿以后所确定的线条集合，足以准确反映所描绘物体的外形轮廓，并将成为最终稿件的一部分。

初学者在没有接受专业培训的条件下，所能接触到的多是完成稿。大量完成稿并不能揭示其绘制过程，初学者难以获知其绘制原理。如果能够通过计算机视觉的方法从绘画成品图像中获得线稿，将对不会绘画或者初来乍到的绘画爱好者有很大帮助。

这是本项目的出发点和归宿。

Photoshop中蕴含的视觉技术

Photoshop本身即是一款通用强大的专业图像绘制合成软件。它已经封装了不计其数的图像处理、视觉处理方法，并提供操作界面与接口给使用者。虽然软件提供的处理手段极为丰富，但并不针对特定需求，需要用户根据情况组合使用。

开发项目前已试验发现，通过合适的特点操作组合，能够在Photoshop中手动实现线稿提取。

本项目将尝试探究并重新实现Photoshop中已被实现的一些技术，并用于构建自动线稿提取的处理工具。

实验目的

从完整的绘画图像中提取轮廓线条集合。

构建一套能实现此功能的图像处理工具。

无论是Photoshop功能探索还是Python代码实现，本项目都力求独创性。

实验内容

- 读取彩色图像
- 执行图像平滑

- 执行图层混合运算
- 执行色阶调整
- 显示合成结果
- 保存结果至文件

开发环境

本实验基于Python 2.7.12实现，用到了以下Python库：

- opencv-python 3.1.0
- numpy 1.12.1
- matplotlib 2.0.2

OpenCV简介

OpenCV是由intel的**Gary Bradsky**在1999年发起，第1版在2000年发布。OpenCV是一个开放的计算视觉库，支持大量的计算视觉、机器学习算法。 目前，OpenCV已经支持主流的编程语言如C++，Python,Java等；并且能够夸平台，在Windows/Linux/OS X/Android/iOS等主要的操作系统上均能够使用。并且提供了CUDA和OPENCL的GPU加速接口。 OpenCV-Python 是OpenCV提供的Python API。它结合了OpenCV C++ API和Python语言两者的优点。

Numpy简介

NumPy的全名为Numeric Python，是一个开源的Python科学计算库，它包括：

- 一个强大的N维数组对象ndrray；
- 比较成熟的（广播）函数库；
- 用于整合C/C++和Fortran代码的工具包；
- 实用的线性代数、傅里叶变换和随机数生成函数

NumPy的优点：

- 对于同样的数值计算任务，使用NumPy要比直接编写Python代码便捷得多；
- NumPy中的数组的存储效率和输入输出性能均远远优于Python中等价的基本数据结构，且其能够提升的性能是与数组中的元素成比例的；
- NumPy的大部分代码都是用C语言写的，其底层算法在设计时就有着优异的性能，这使得NumPy比纯Python代码高效得多。

Matplotlib简介

matplotlib是基于numpy的一套Python工具包。这个包提供了丰富的数据绘图工具，主要用于绘制一些统计图形。其中matplotlib.pyplot是一些命令行风格函数的集合，使matplotlib以类似于MATLAB的方式工作。每个pyplot函数对一幅图片(figure)做一些改动：比如创建新图片，在图片创建一个新的作图区域(plotting area)，在一个作图区域内画直线，给图添加标签(label)等。

实验步骤

准备工作

引入需使用的库：

```
from __future__ import print_function
import numpy as np
import cv2
import matplotlib.pyplot as plt
from time import clock
```

程序运行耗时计算：

```
before_process = clock() # 置于运算前
```

```
after_process = clock() # 置于运算后
plt.figure(num='anime' + ' time elapsed: %.3f s' %
               (after_process - before_process)) # 窗口标题显示运行耗时
```

读取图片：

```
origin = cv2.imread(filename, cv2.IMREAD_COLOR)
```

`cv2.imread()`的第一个参数为图片文件名，第二个参数可取以下值：

```
cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the
default flag.
cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode
cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel
```

本实验需要用到彩色图像，故选择`cv2.IMREAD_COLOR`。

`cv2.imread()`读取到的彩色图像包含3个通道，默认按BGR格式存储而非通用的RGB格式，故作格式转换：

```
b, g, r = cv2.split(origin)
origin = cv2.merge([r, g, b])
```

平滑处理

- 双向滤波

由于常用的几种网络图像格式中，只有`png`格式质量相对有保证，其他图像格式中包含了明显的噪声，故强制对原图像添加`双向滤波`平滑处理。

```
    if get_file_extension(filename) != 'png':
            origin = cv2.bilateralFilter(origin,
                                         5, # d – Diameter of each pixel neighborhood that is
used during filtering.
                                         80, # Filter sigma in the color space.
                                         120) # Filter sigma in the coordinate space.
```

此处几个参数值是根据文档说明与样本图片反复测试推测出的较理想值。

- 高斯滤波

由于输入的图像质量不高，锯齿感较为明显，噪声较多，故添加图像平滑作为前提运算。平滑方法为`高斯滤波`。

```
    origin = cv2.GaussianBlur(origin,
                              (0, 0), # Gaussian kernel size. (width,height). They can be zero's
and  computed from sigma
                              0.8) # sigmaX – Gaussian kernel standard deviation in X direction.
                                # sigmaY – Gaussian kernel standard deviation in Y direction; 0
by default
                                # if sigmaY is zero, it is set to be equal to sigmaX,
```

此处的参数`sigmaX=sigmaY=0.8`并不是随意设置的，而是在Photoshop测试多张样本图片后得出的较理想值。

Photoshop中的高斯模糊方法操作较为简单，只提供半径(sigma)一个可调整参数。在Photoshop中执行半径0.8的高斯模糊可得到完全相同的效果。

图像混合

- 颜色反相

根据Photoshop中得出的操作经验，图层混合运算前，需先对原图像进行反色操作。

```
def my_inv(src):
    width, height, ch = src.shape
    res = src.copy()
    for i in range(width):
        for j in range(height):
            res[i, j] = (255 - src[i, j][0], 255 - src[i, j][1], 255 - src[i, j][2])
    return res
```

也可调用`opencv`库中提供的反相方法：

```
    inv = origin.copy()
    inv = cv2.bitwise_not(inv)
```

原图像经过`反色+高斯滤波`后命名为`Inverse && GaussianBlur`在窗口中显示。

- 图层混合

Photoshop中各式各样的混合模式，软件中未给出原理，但其计算公式都可查阅得到。

Photoshop中`颜色减淡`图层混合模式的实现：

```
def my_merge(src, append):
    res = src.copy()
    width, height, ch = src.shape
    for i in range(width):
        for j in range(height):
            for c in range(ch):
                base = src[i, j][c]
                overlay = append[i, j][c]
                # Photoshop混合模式之颜色减淡
                if overlay < 255:
                    res[i, j][c] = min((base << 8) // (255 - overlay), 255)
                else:
                    res[i, j][c] = 255
    return res
```

Photoshop中`变亮`图层混合模式的实现：

```
def my_merge2(src, append):
    res = src.copy()
    width, height, ch = src.shape
    for i in range(width):
        for j in range(height):
            for c in range(ch):
                base = src[i, j][c]
                overlay = append[i, j][c]
                # Photoshop混合模式之变亮
                res[i, j][c] = max(base, overlay)
    return res
```

将原图像与`Inverse && GaussianBlur`分别视作图层，进行混合，混合后的图像（图层）命名为`Merged`。

```
merge = my_merge(origin, blur) # original在下，Inverse && GaussianBlur在上，以颜色减淡方式混合
```

一个无意之中获得的重大发现

由于编写代码过程中曾经写错过计算公式，混合图层叠放次序被颠倒，出现了截然不同的混合结果。但此混合效果非同一般，对最终结果有关键影响。原本图层混合操作到上一步就截止了，现在得以灵活变化的方式继续进行。

逆序的混合图像命名为`Merged-Reverse`。

```
merge_r = my_merge(blur, origin) # Inverse && GaussianBlur在下，original在上，以颜色减淡方式混合
```

上述`Merged`与`Merged-Reverse`已是提取到的线稿集了，区别是各有侧重点，又不太充分。再对这两份不同风格的线稿集进行`变亮`混合，命名为`Merged-2p`。

`变亮`的实质是在两图层相同位置像素上取较大值。

```
merge_2p = my_merge2(merge, merge_r) # Merged在下，Merged-Reversel在上，以变亮方式混合
```

色阶调整

以上`Merged-2p`显得泛白、太亮，有必要做修正以突出线条主题，强调细节。比起简易的亮度对比度调整，色阶调整的针对性更强。

要实现色阶调整，先生成颜色值/灰度值映射表，再转换颜色。

```
def createRGBColorTable(Highlight, Midtones, Shadow, OutputHighlight, OutputShadow):
    diff = int(Highlight - Shadow)
    outDiff = int(OutputHighlight - OutputShadow)

    if not((Highlight <= 255 and diff <= 255 and diff >= 2) or
            (OutputShadow <= 255 and OutputHighlight <= 255 and outDiff < 255) or
            (not(Midtones > 9.99 and Midtones > 0.1) and Midtones != 1.0)):
        raise Exception('Invalid level parameters!')

    coef = 255.0 / diff
    outCoef = outDiff / 255.0
    exponent = 1.0 / Midtones

    colorTable = [0] * 256

    for i in range(0, 256):
        colorTable[i] = i
        # calculate black field and white field of input level
        if colorTable[i] <= Shadow:
            v = 0
        else:
            v = int((colorTable[i] - Shadow) * coef + 0.5)
            if v > 255:
                v = 255
        # calculate midtone field of input level
        v = int(pow(v / 255.0, exponent) * 255.0 + 0.5)
        # calculate output level
        colorTable[i] = int(v * outCoef + OutputShadow + 0.5)

    return colorTable
```

```
def adjustRGBLevel(src, Highlight, Midtones, Shadow, OutputHighlight, OutputShadow):
    width, height, channels = src.shape
    dst = np.zeros(src.shape, np.uint8)

    try:
        colorTable = createRGBColorTable(
            Highlight, Midtones, Shadow, OutputHighlight, OutputShadow)
    except:
        raise Exception('Invalid colorTable!')

    for i in range(width):
        for j in range(height):
            for c in range(channels):
                dst[i, j][c] = colorTable[src[i, j][c]]

    return dst
```

色阶调整后输出命名为`Merged-2p+`。

```
merge_2pa = adjustRGBLevel(merge_2p,
    255, # 高光输入
    0.8, # 中间调输入
    128, # 阴影输入
    255, # 高光输出
    0) # 阴影输出
```

所取的各参数值是在Photoshop测试多张样本图片后得出的较理想值，均与软件功能直接对接，具有实际意义。

显示图像

将以上各步所得输出，加上小标题后显示在窗口中：

```
    plt.subplot(2, 4, 1)
    plt.title('Orinigal image')
    plt.imshow(origin)
    plt.axis('off')  # 不显示坐标尺寸

# ...
```

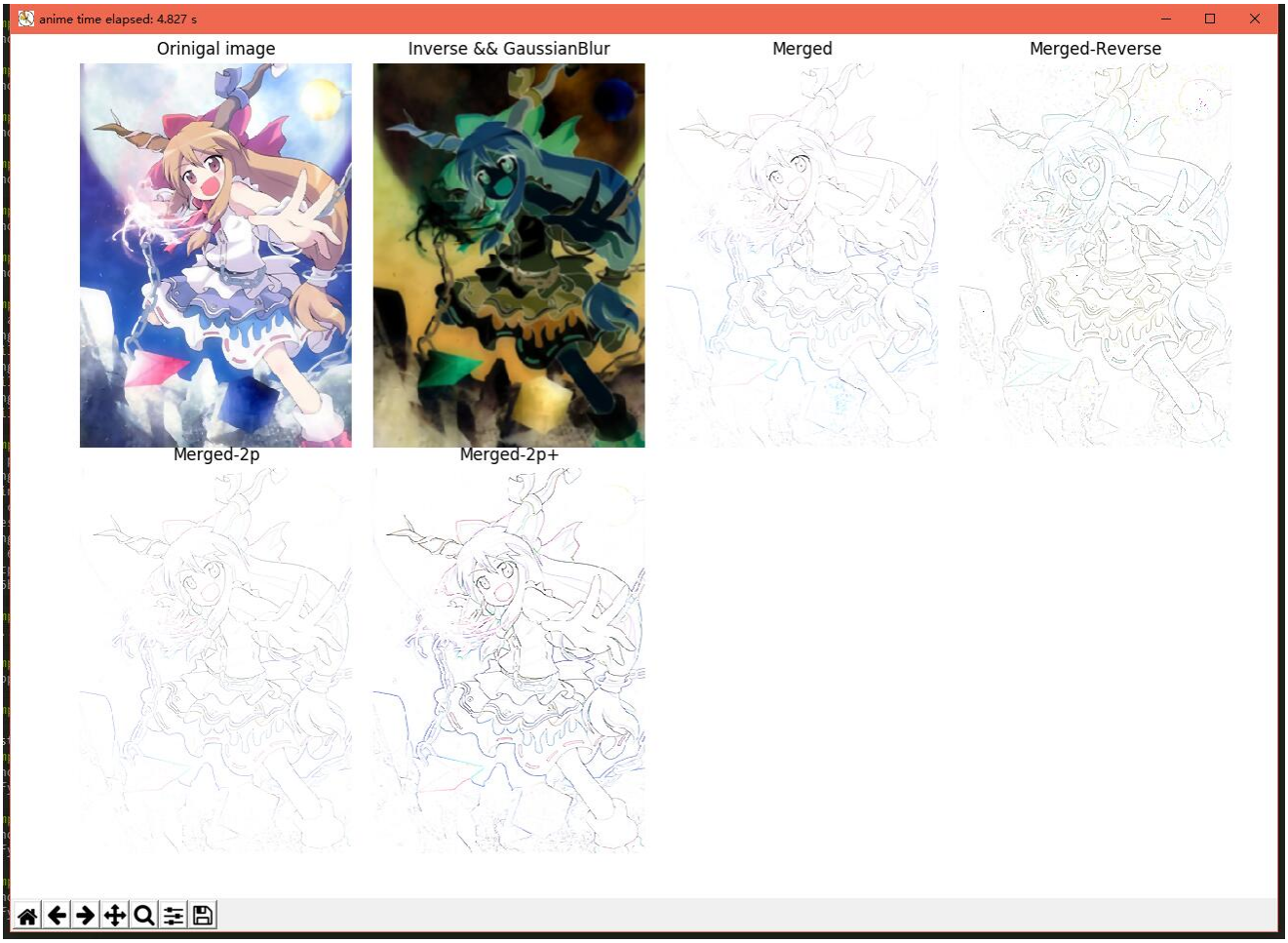
保存图像

将以上各步所得输出，以原图像的格式保存到文件。`cv2.imwrite()`同样只支持BGR通道，故需对RGB通道进行重组。

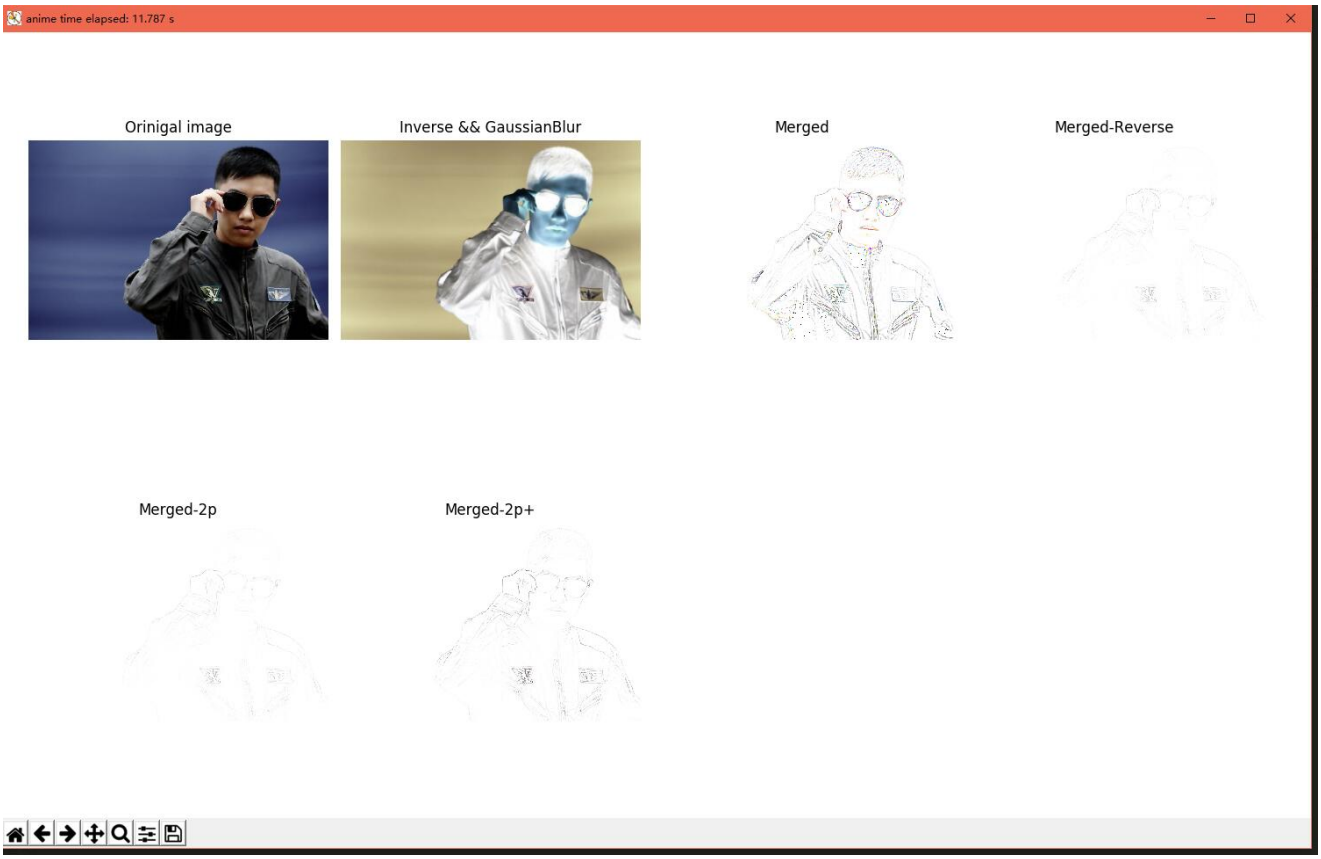
```
    r, g, b = cv2.split(merge)
    merge = cv2.merge([b, g, r])
    cv2.imwrite(data_dir + get_file_name(filename) +
                '.' + get_file_extension(filename), merge)

# ...
```

实验结果

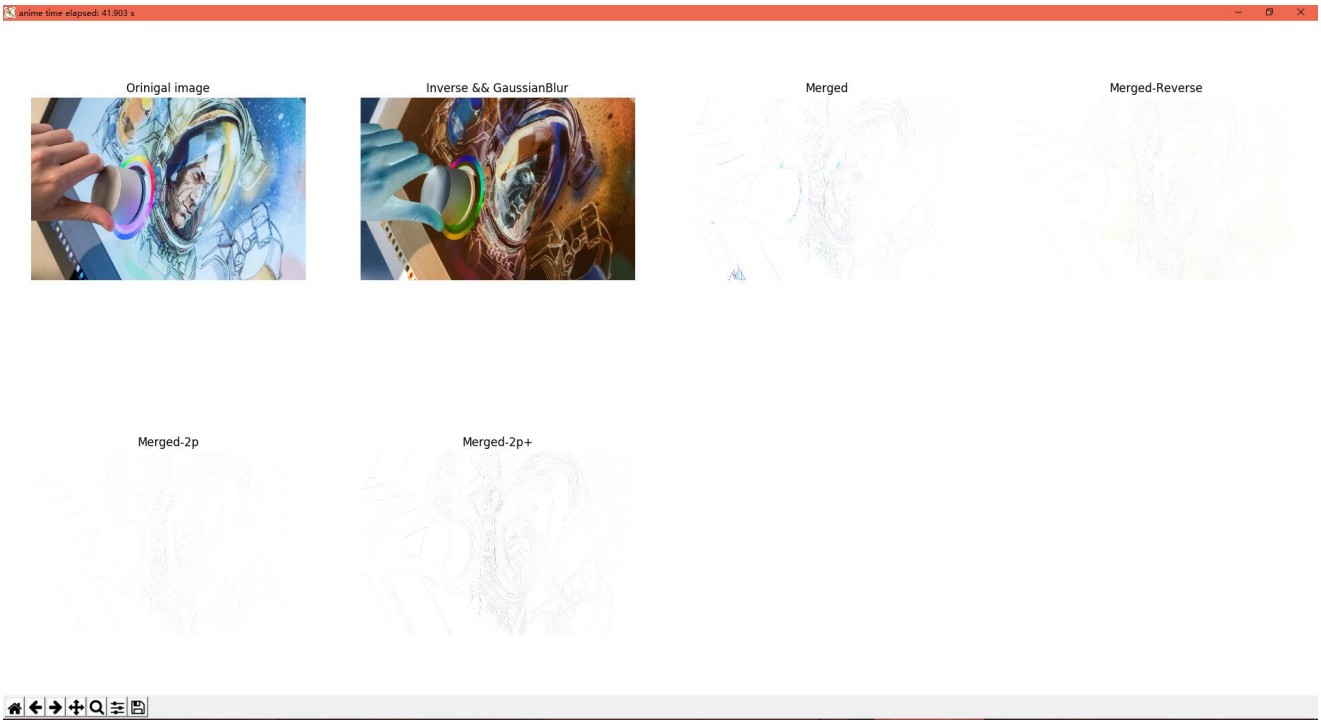
- 绘画图片测试


anime time elapsed: 4.827 s
Orinigal image
Inverse && GaussianBlur
Merged
Merged-Reverse
Merged-2p
Merged-2p+

- 真人照片测试


anime time elapsed: 11.787 s
Orinigal image
Inverse && GaussianBlur
Merged
Merged-Reverse
Merged-2p
Merged-2p+

- 照片与绘画混合测试



实验总结

可以看出，本项目对绘画作品，尤其是存在大面积连续色块的图像效果良好；对于实拍照片，由于细节变化频繁，很可能会失效。

图层混合时，由于直接遍历、对每个像素点操作，时间复杂度较高，遇到高分辨率图像时处理时间明显延长，远远不能追上高效专业的Photoshop软件。

处理步骤中存在的一个明显缺陷，在于`变亮`混合操作。实际图像中“应该被保留”与“不能被保留”不能一刀切地做决定，需要区分对待，可能需要人工操作；这种复杂的工作要交给机器执行，势必要依赖机器学习。

如果引入神经网络，或能替换传统方法，使处理效果大有改观，且能应对更为复杂多变的情况。但是欲训练由图像生成图像的神经网络，训练难度较大，受本人水平所限，短期内暂不能实现。不过未来长期的努力应能解决这一难题。

参考文献

- http://avnpc.com/pages/photoshop-layer-blending-algorithm
- http://blog.csdn.net/geyalu/article/details/50190121
- http://lib.csdn.net/article/opencv/26910
- https://www.douban.com/note/293097555/
- http://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html?highlight=gaussianblur