学习笔记

导航

本文使用的模板地址:https://www.overleaf.com/latex/templates/morelull-sample/sfrmdxnrnbbn

推荐一个 Emacs Org Mode 模板：用纯文本组织你的生活，可以用于**制定待办事项列表、制定学习计划、写博客、涂鸦文档 & 写论文**。

1 图像相似度量

1.1 直方图

运行速度快

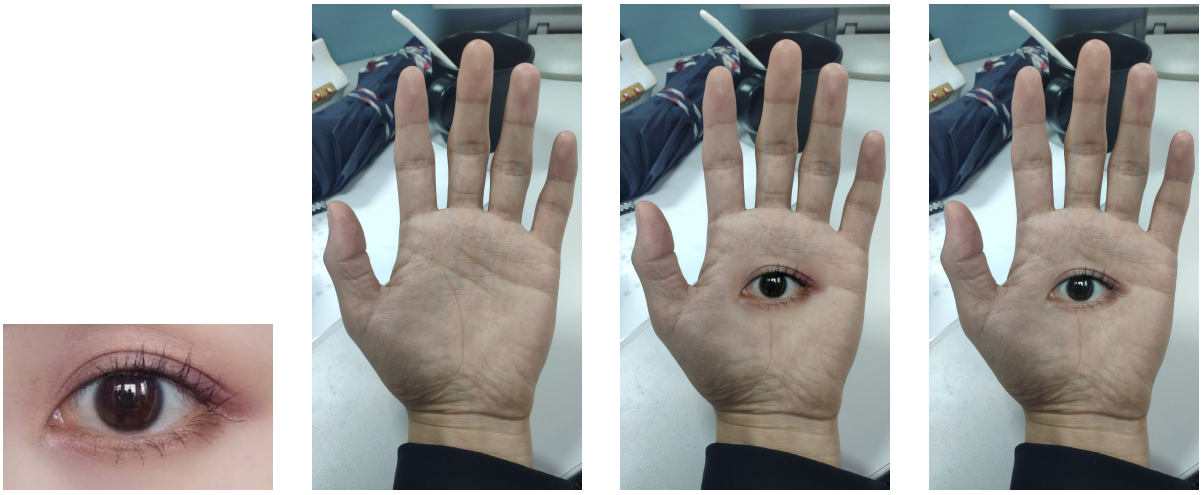
```
    def compare_img_hist(img1, img2):
    """
    Compare the similarity of two pictures using histogram(直方图)
    Attention: this is a comparision of similarity, using histogram to calculate

    For example:
    1. img1 and img2 are both 720P .PNG file, and if compare with img1, img2 only add a black dot(about 9*9px)
     , the result will be 0.999999999953

    :param img1: img1 in MAT format(img1 = cv2.imread(image1))
    :param img2: img2 in MAT format(img2 = cv2.imread(image2))
    :return: the similarity of two pictures
    """
    # Get the histogram data of image 1, then using normalize the picture for better compare
    img1_hist = cv.calcHist([img1], [1], None, [256], [0, 256])
    img1_hist = cv.normalize(img1_hist, img1_hist, 0, 1, cv.NORM_MINMAX, -1)

    img2_hist = cv.calcHist([img2], [1], None, [256], [0, 256])
    img2_hist = cv.normalize(img2_hist, img2_hist, 0, 1, cv.NORM_MINMAX, -1)

    similarity = cv.compareHist(img1_hist, img2_hist, 0)

    return similarity
```

2 图像融合

- 泊松融合；利用偏微分方程实现不同图像上**不同区域**的**无缝融合**【2003 年发表 [?]】;

```
import cv2
import numpy as np
obj = cv2.imread("obj.jpg")
im = cv2.imread("im.jpg")
# 创建mask, 绘制多边形 ***
mask = np.zeros(obj.shape, obj.dtype)
point = edge(obj) # 二维数组
poly = np.array(point, np.int32)
cv2.fillPoly(mask, [poly], (255, 255, 255))
# 待融合位置的坐标
center = (x, y)
```

```
# options cv2.MIXED_CLONE/cv2.NORMAL_CLONE/cv2.MONOCHROME_TRANSFER
output = cv2.seamlessClone(obj, im, mask, center, cv2.NORMAL_CLONE)
```



(a) (b) (c) (d)

图 1: 泊松融合。(a)表示目标图像；(b)表示背景图片；(c)表示 mask 为矩形框的融合图片；(d)表示 mask 为目标边缘坐标的融合图片

可以看到，由于目标图像中存在过多的无用边界信息，导致融合后的视觉差较大（图1(c)）。事实上，最好的做法是将需要的 ROI 抠出来。那么，**如何获取目标图像准确的 ROI 呢？**

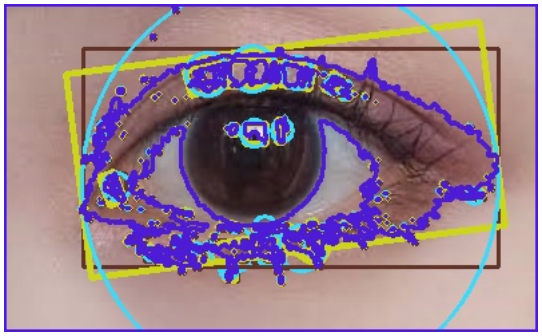


图 2: 目标轮廓

图2为不同的边缘检测结果。然而，越精细的轮廓结构往往意味着数据分布的噪声程度越高。长话短说，**我现在要蓝色数据点的外围轮廓**。如果觉得太严格了，可以退而求其次，取黄色方框区域为 ROI。

```
import numpy as np
from scipy.spatial import ConvexHull
import matplotlib.pyplot as plt
def min_contour(points):
    hull = ConvexHull(points)  ###计算外接凸图案的边界点
    plt.plot(points[:, 0], points[:, 1], 'o')
```

```
        plt.plot(points[hull1, 0], points[hull1, 1], 'r--^', lw=2)
        alpha_shape = alphashape.alphashape(points, 3) # 这里的3是alpha vlue, 数字越大, 拟合效果会越好,
            耗时也会久一些
        fig, ax = plt.subplots()
        ax.scatter(*zip(*points))
        ax.add_patch(PolygonPatch(alpha_shape, alpha=0.3))
        plt.show()
        return np.vstack([points[hull1, 0], points[hull1, 1]]).T
```

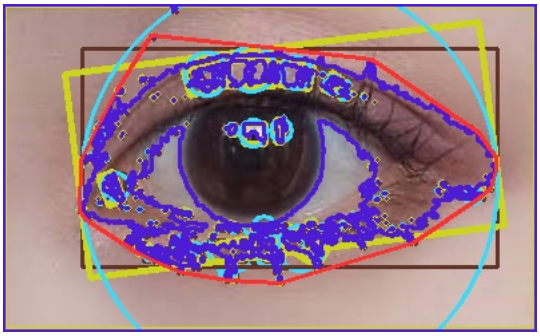


图 3: 红色框为目标体的最小外接多边形【还可以继续拟合】

```
def edge(img):
    img = cv2.imread(img, cv2.IMREAD_UNCHANGED)
    H, W = img.shape[:2]
    ret, thresh = cv2.threshold(cv2.cvtColor(img.copy(), cv2.COLOR_BGR2GRAY), 127, 255, cv2.
        THRESH_BINARY)
    # findContours函数查找图像里的图形轮廓
    # 函数参数thresh是图像对象
    # 层次类型, 参数cv2.RETR_EXTERNAL是获取最外层轮廓, cv2.RETR_TREE是获取轮廓的整体结构
    # 轮廓逼近方法
    # 输出的返回值, contours是图像的轮廓、hier是层次类型
    contours, hier = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    temp = [[0, 0], [0, H], [W, 0], [H, W]]
    for i in contours: point.extend(i[0].tolist())
    for i in temp:
        if i in point: point.remove(i)

    for c in contours:
        # 轮廓绘制方法一
        # boundingRect函数计算边框值, x, y是坐标值, w, h是矩形的宽和高
        x, y, w, h = cv2.boundingRect(c)
        # 在img图像画出矩形, (x, y), (x + w, y + h)是矩形坐标, (0, 255, 0)设置通道颜色, 2是设置线条粗
            度
        cv2.rectangle(img, (x, y), (x + w, y + h), (41, 52, 98), 2)

        # 轮廓绘制方法二
        # 查找最小区域
        rect = cv2.minAreaRect(c)
        # 计算最小面积矩形的坐标
        box = cv2.boxPoints(rect)
        # 将坐标规范化为整数
```

```
            box = np.int0(box)
            # 绘制矩形
            cv2.drawContours(img, [box], 0, (28, 214, 206), 3)

            # 轮廓绘制方法三
            # 圆心坐标和半径的计算
            (x, y), radius = cv2.minEnclosingCircle(c)
            # 规范化为整数
            center = (int(x), int(y))
            radius = int(radius)
            # 勾画圆形区域
            img = cv2.circle(img, center, radius, (254, 219, 57), 2)
        ## 轮廓绘制方法四
        # 围绕图形勾画蓝色线条
        cv2.drawContours(img, contours, -1, (214, 28, 78), 2)
        point = min_contour(point)
        contours = []
        for i in point: contours.append([i])
        cv2.polylines(img, [np.array(contours)], True, (48, 48, 255), 2)

        cv2.imwrite("edge.jpg", img)
        cv2.imshow("contours", img)
        cv2.waitKey()
        cv2.destroyAllWindows()
        return point
```

- 深度学习方法

3 RGB&HSV

3.1 RGB 的局限性

RGB 颜色模型是利用三个颜色分量来表示任何颜色。

- RGB 颜色模型对亮度比较敏感。在自然环境下获取到的图像易受自然光照、遮挡和阴影等情况的影响，而 RGB 的三个分量又与亮度密切相关
- 人眼对 RGB 颜色分量的敏感度不一样【对红色最不敏感，对蓝色最敏感】
- 难推测出精确的颜色分量数值。黄色在 RGB 中为（255, 255, 0），而在 HSV 中，黄色仅由一个值决定 Hue = 60

结论：RGB 颜色模型更适合显示系统

3.2 HSV 颜色模型

HSV 颜色模型组成：

- Hue（0 - 360）：色调、色相

- Saturation（0-100%）：饱和度、色彩纯净度
- Value（0-100%）：明度

Hue 用角度表示色彩信息，如图，Hue = 0 表示红色，Hue = 120 表示绿色，Hue = 240 表示蓝色等等。

当 Hue = 60 时，如图水平方向为饱和度，饱和度越高，颜色越深，越接近光谱色的程度；饱和度越低，越接近白色。竖直方向为明度，明度越高，颜色越亮，明度越低，越接近黑色。

可以看到，HSV 对于用户来说是一种比较直观的颜色模型，可以轻松得到单一颜色【指定颜色角 H，并让 $V = S = 1$】，通过对 S 和 V 拉伸即可得到不同深浅的颜色。

4 基于锚点的检测框合成方法

Algorithm 1: 基于锚点的检测框合成方法

输入： ROI 区域；最大搜索步数 N

初始化： 锚点 $\mathcal{P}$；搜索步数 $n = 0$

while $n \leq N$ **do**

步骤一，定义相似 $\mathcal{F}$，相似度阈值 ϵ；

步骤二，以 $\mathcal{P}$ 为中心生成 Base Patches，并基于 $\mathcal{F}$ 和 ϵ 在 ROI 区域，搜索相似块 $\mathcal{P}'$；

步骤三，整合 $\mathcal{P}'$，生成检测框 D；

$n = n + 1$

end

输出： D

- 要点 1：定义相似 $\mathcal{F}$
- 要点 2：优化搜索

5 交互式标注脚本

```
# -*-coding: utf-8 -*-

import cv2,  os
import numpy as np
from typing import Callable
from pybaseutils import image_utils
import os.path as osp

class DrawImageMouse(object):
    """使用鼠标绘图"""

    def __init__(self, max_point=-1, line_color=(0, 0, 255), text_color=(255, 0, 0), thickness=2):
        """
␣␣␣␣␣␣␣␣␣␣␣␣:param␣max_point:␣最多绘图的点数，超过后将绘制无效；默认-1表示无限制
␣␣␣␣␣␣␣␣␣␣␣␣:param␣line_color:␣线条的颜色
␣␣␣␣␣␣␣␣␣␣␣␣:param␣text_color:␣文本的颜色
```

```
        :param thickness: 线条粗细
        """
        self.max_point = max_point
        self.line_color = line_color
        self.text_color = text_color
        self.focus_color = (0, 255, 0)  # 鼠标焦点的颜色
        self.thickness = thickness
        self.key = -1  # 键盘值
        self.orig = None  # 原始图像
        self.last = None  # 上一帧
        self.next = None  # 下一帧或当前帧
        self.polygons = np.zeros(shape=(0, 2), dtype=np.int32)  # 鼠标绘制点集合

    def clear(self):
        self.key = -1
        self.polygons = np.zeros(shape=(0, 2), dtype=np.int32)
        if self.orig is not None: self.last = self.orig.copy()
        if self.orig is not None: self.next = self.orig.copy()

    def get_polygons(self):
        """获得多边形数据"""
        return self.polygons

    def task(self, image, callback: Callable, winname="winname"):
        """
        鼠标监听任务
        :param image: 图像
        :param callback: 鼠标回调函数
        :param winname: 窗口名称
        :return:
        """
        self.orig = image.copy()
        self.last = image.copy()
        self.next = image.copy()
        cv2.namedWindow(winname, flags=cv2.WINDOW_NORMAL)
        cv2.setMouseCallback(winname, callback, param={"winname": winname})
        while True:
            self.key = self.show_image(winname, self.next, delay=25)
            if (self.key == 13) and len(self.polygons) > 0:  # 按回车键13表示完成绘制
                break
            elif self.key == 27:  # 按ESC退出程序
                exit(0)
            elif self.key == 99:  # 按键盘c重新绘制
                self.clear()
        cv2.setMouseCallback(winname, self.event_default)

    def event_default(self, event, x, y, flags, param):
        pass

    def event_draw_rectangle(self, event, x, y, flags, param):
        """绘制矩形框"""
        if len(self.polygons) == 0: self.polygons = np.zeros(shape=(2, 2), dtype=np.int32)  # 多边形轮廓
        point = (x, y)
        if event == cv2.EVENT_LBUTTONDOWN:  # 左键点击,则在原图打点
```

```
            self.next = self.last.copy()
            self.polygons[0, :] = point
            cv2.circle(self.next, point, radius=5, color=self.focus_color, thickness=self.thickness)
        elif event == cv2.EVENT_MOUSEMOVE and (flags & cv2.EVENT_FLAG_LBUTTON): # 按住左键拖曳，画框
            self.next = self.last.copy()
            cv2.circle(self.next, self.polygons[0, :], radius=4, color=self.focus_color, thickness=self.
                thickness)
            cv2.circle(self.next, point, radius=4, color=self.focus_color, thickness=self.thickness)
            cv2.rectangle(self.next, self.polygons[0, :], point, color=self.line_color, thickness=self.
                thickness)
        elif event == cv2.EVENT_LBUTTONUP: # 左键释放，显示
            self.next = self.last.copy()
            self.polygons[1, :] = point
            cv2.rectangle(self.next, self.polygons[0, :], point, color=self.line_color, thickness=self.
                thickness)

    def event_draw_polygon(self, event, x, y, flags, param):
        """绘制多边形"""
        exceed = self.max_point > 0 and len(self.polygons) >= self.max_point
        self.next = self.last.copy()
        point = (x, y)
        text = str(len(self.polygons))
        if event == cv2.EVENT_LBUTTONDOWN: # 左键点击,则在原图打点
            cv2.circle(self.next, point, radius=5, color=self.focus_color, thickness=self.thickness)
            cv2.putText(self.next, text, point, cv2.FONT_HERSHEY_SIMPLEX, 0.5, self.text_color, 2)
            if len(self.polygons) > 0:
                cv2.line(self.next, self.polygons[-1, :], point, color=self.line_color, thickness=self.
                    thickness)
            if not exceed:
                self.last = self.next
                self.polygons = np.concatenate([self.polygons, np.array(point).reshape(1, 2)])
        else:
            cv2.circle(self.next, point, radius=5, color=self.focus_color, thickness=self.thickness)
            if len(self.polygons) > 0:
                cv2.line(self.next, self.polygons[-1, :], point, color=self.line_color, thickness=self.
                    thickness)

    @staticmethod
    def polygons2box(polygons):
        """将多边形转换为矩形框"""
        xmin = min(polygons[:, 0])
        ymin = min(polygons[:, 1])
        xmax = max(polygons[:, 0])
        ymax = max(polygons[:, 1])
        return [xmin, ymin, xmax, ymax]

    def show_image(self, title, image, delay=5):
        """显示图像"""
        cv2.imshow(title, image)
        key = cv2.waitKey(delay=delay) if delay >= 0 else -1
        return key

    def draw_image_rectangle_on_mouse(self, image, winname="draw_rectangle"):
        """
```

```
            获得鼠标绘制的矩形框box=[xmin,ymin,xmax,ymax]
            :param image:
            :param winname:窗口名称
            :return:box is [xmin,ymin,xmax,ymax]
            """
            self.task(image, callback=self.event_draw_rectangle, winname=winname)
            polygons = self.get_polygons()
            box = self.polygons2box(polygons)
            return box

        def draw_image_polygon_on_mouse(self, image, winname="draw_polygon"):
            """
            获得鼠标绘制的矩形框box=[xmin,ymin,xmax,ymax]
            :param image:
            :param winname:窗口名称
            :return:polygons is (N,2)
            """
            self.task(image, callback=self.event_draw_polygon, winname=winname)
            polygons = self.get_polygons()
            return polygons

    def draw_image_rectangle_on_mouse_example(root, img, winname="draw_rectangle"):
        """
        获得鼠标绘制的矩形框
        :param root:
        :param img:
        :param winname:窗口名称
        :return:box=[xmin,ymin,xmax,ymax]
        """
        image = cv2.imread(osp.join(root, img))
        # 通过鼠标绘制矩形框rect
        mouse = DrawImageMouse()
        box = mouse.draw_image_rectangle_on_mouse(image, winname=winname)
        # 裁剪矩形区域,并绘制最终的矩形框
        roi: np.ndarray = image[box[1]:box[3], box[0]:box[2]]
        if roi.size > 0: mouse.show_image("Image_ROI", roi)
        floder = input("请输入输入类别【0: 积水\n1: 晴天强光\n2: 晴天弱光\n3: 雨天强光\n4: 雨天弱光\n5:
            花屏\n6: 背景\n】: ")
        path_floder = osp.join(root, floder)
        if not osp.exists(path_floder): os.makedirs(path_floder)
        cv2.imwrite(osp.join(path_floder, img), roi)
        image = image_utils.draw_image_boxes(image, [box], color=(0, 0, 255), thickness=1)
        mouse.show_image(winname, image, delay=0)
        return box

    def draw_image_polygon_on_mouse_example(root, img, winname="draw_polygon"):
        """
        获得鼠标绘制的多边形
        :param root:
        :param img:
        :param winname:窗口名称
        :return:polygons is (N,2)
        """
        image = cv2.imread(osp.join(root, img))
```

```
        # 通过鼠标绘制多边形
        mouse = DrawImageMouse(max_point=-1)
        polygons = mouse.draw_image_polygon_on_mouse(image, winname=winname)
        image = image_utils.draw_image_points_lines(image, polygons, thickness=2)
        mouse.show_image(winname, image, delay=0)
        return polygons

    if __name__ == '__main__':
        root = "D:/Data_cug/File/pic/"
        fw = open(root + "crop.txt", "w+")
        img = "dog.jpg"
        # 绘制矩形框
        out = draw_image_rectangle_on_mouse_example(root, img)
        # 绘制多边形
        # out = draw_image_polygon_on_mouse_example(image_file)
        fw.write(img + ',' + ','.join([str(i) for i in out]) + '\n')
        fw.close()
```

6 Fork 是什么？

6.1 简介

Fork 是一款面向 GitHub 开发者的 Git 客户端，它提供了直观的用户界面和实用的工具，支持多个仓库的管理，可以轻松地切换分支、合并代码和解决冲突。它还提供了实时的代码差异对比、代码提交历史记录、分支可视化等功能，让开发者可以更加直观地了解代码的变化和演进。此外，Fork 还支持任务列表、代码注释、代码搜索等实用功能，方便开发者进行项目管理和协作。

总的来说，Fork 是一款易于上手（尤其对于非开发人员）且功能强大的版本管理软件

6.2