Experiment 12

517030910356 谢知晖

目录

1 实验准备

1.1 实验环境

本次实验环境为 Windows 平台下的 Python(版本号 3.6)。
另外，实验中还用到的 Python 库有:

1. OpenCV(版本号 3.4.3)
2. NumPy(版本号 1.15.4)

1.2 实验目的

在前一次实验中，我们已经实现了边缘的检测，对于图像的特征提取有了初步的认识。但是我们还没有解决的问题是：提取出特征之后，怎么把它运用到具体的应用，图像识别当中呢?

于是，本次实验我们通过实现另外一种特征提取"SIFT" 来进一步理解图像识别。由于 SIFT 特征具有诸如旋转不变形等良好性质，我们能够通过这次实验来学习更好的特征提取方法，这对于我们图像处理方面的知识无疑是有很大提升的。

1.3 实验原理

针对本次实验的核心内容:SIFT 算子，我将分步介绍其原理。

1.3.1 检测尺度空间极值点

在这个阶段，主要是侦测兴趣点，也就是 SIFT 架构中的关键点。影像在不同的尺度下用高斯滤波器 (Gaussian filters) 进行卷积 (convolved)，然后利用连续高斯模糊化影像差异来找出关键点。关键点是根据不同尺度下的高斯差 (Difference of Gaussians,DoG) 的最大最小值。也就是说，DoG 影像的 $D(x, y, \sigma)$ 是由:

$$D(x, y, \sigma) = L\left(x, y, k_i\sigma\right) - L\left(x, y, k_j\sigma\right) \tag{1}$$

得到的，其中 $L(x, y, k\sigma)$ 是在尺度 $k\sigma$ 的条件下，由原始影像 $I(x, y)$ 与高斯模糊 $G(x, y, k\sigma)$ 进行卷积，例如:

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y) \tag{2}$$

其中 $G(x, y, k\sigma)$ 是尺度可变高斯函数 $G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(x^2+y^2\right)/2\sigma^2}$。

由上式可知 DoG 影像是原始影像与不同尺度倍率 $k_i\sigma$、$k_j\sigma$ 的高斯模糊后之差值。SIFT 算法为了求得在不同尺度倍率之下 DoG 影像的极大值，先将原始影像与不同尺度倍率的高斯模糊进行卷积，这些经高斯模糊处理后的影像依其尺度倍率以 2 倍为一单位分组，并且 $k_i\sigma$ 通常为一个选定后的定值，因此在每一组内经高斯模糊处理后的影像数量相同，此时将同一组相邻的经高斯模糊处理后的影像两两相减可得其 DoG 影像。

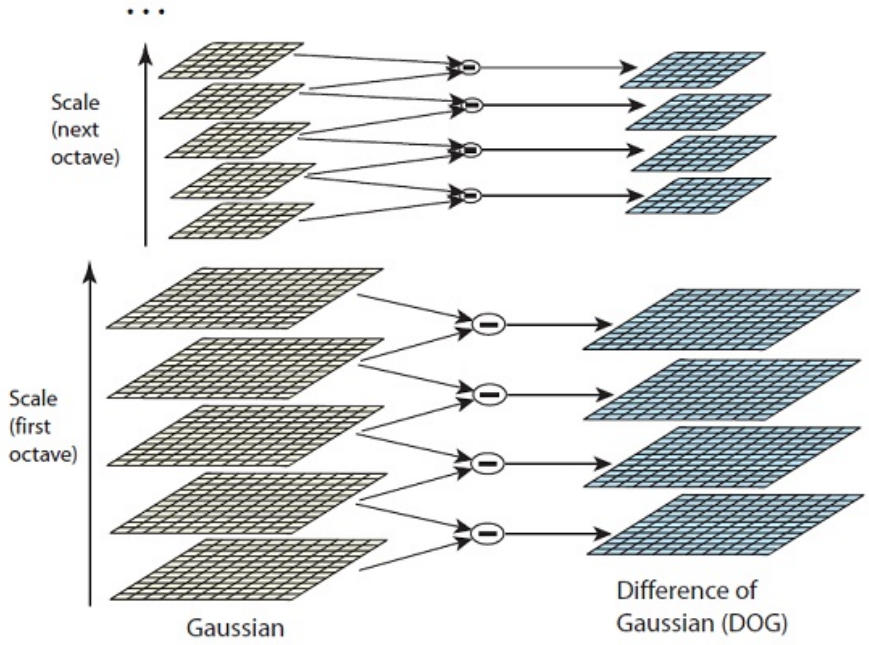

Fig1:DoG

一旦得到 DoG 影像后，可找出 DoG 影像中的极大、极小值作为关键点。为了决定关键点，DoG 影像中的每个像素会跟以自己为中心周围的八个像素，以及在同一组 DoG 影像中相邻尺度倍率相同位置的九个像素作，一共二十六个点作比较，若此像素为这二十六个像素中的最大、最小值，则此称此像素为关键点。

SIFT 算法中关键点的侦测是一种斑点检测 (Blob detection) 的一种变形，也就是使用拉普拉斯算子来求出各个倍率及空间中的最大值。高斯差可近似为拉普拉斯算子运算后的结果，因建立高斯金字塔的过程是一种尺寸正规化拉普拉斯运算的近似。

1.3.2 精确定位极值点

在不同尺寸空间下可能找出过多的关键点，有些关键点可能相对不易辨识或易受噪声干扰。SIFT 算法的下一步将会借由关键点附近像素的信息、关键点的尺寸、关键点的主曲率来定位各个关键点，借此消除位于边上或是易受噪声干扰的关键点。

邻近信息插补 对于各个可能的关键点，插补邻近信息可决定其位置。

计算极值得差补位置更能够提供配对的准确度及可靠性，此方法使用 DoG 影像 $D(x, y, \sigma)$ 的二次泰勒级数，并以关键点作为原点，其展开式可写成:

$$D(\mathrm{x}) = D + \frac{\partial D^T}{\partial \mathrm{x}}\mathrm{x} + \frac{1}{2}\mathrm{x}^T\frac{\partial^2 D}{\partial \mathrm{x}^2}\mathrm{x} \tag{3}$$

其中 D 与其偏微分的值由关键点的位置决定，变数 $\mathrm{x} = (x, y, \sigma)$ 则是到此关键点的偏移量。

将上式对 x 微分后设为零，便可求出极值 $\hat{\mathrm{X}}$ 的位置。若 $\hat{\mathrm{X}}$ 的任一项参数大于 0.5，代表此时有另一关键点更接近极值，将舍弃此关键点并对新的点作插补。反之若所有参数皆小于 0.5，此时将偏移量加回关键点中找出极值的位置。

舍弃不明显关键点 此步骤将计算上述二次泰勒级数 $D(\mathrm{x})$ 在 $\hat{\mathrm{X}}$ 的值。若此值小于 0.03，则舍弃此关键点。反之保留此关键点，并记录其位置为 $\mathrm{y} + \hat{\mathrm{x}}$，其中 y 是一开始关键点的位置。

消除边缘响应 DoG 函数对于侦测边缘上的点相当敏感，即使其找出位于边缘的关键点易受噪声干扰也会被侦测为关键点。因此为了增加关键点的可靠性，需要消除有高度边缘响应但其位置不佳不符合需求的关键点。

为了找出边缘上的点，可分别计算关键点的主曲率，对于在边上的关键点而言，因为穿过边缘方向的主曲率会远大于沿着边缘方向上的主曲率，因此其主曲率比值远大于位于角落的比值。

为了算出关键点的主曲率，可解其二次 Hessian 矩阵矩阵的特征值:

$$\mathrm{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{4}$$

其特征值的比例与特征点主曲率的比例相同，因此假设解出的特征值为 α、β，其中 $\alpha > \beta$。在上述矩阵中:

$$\begin{cases} \mathrm{Tr}(\mathrm{H}) = D_{xx} + D_{yy} = \alpha + \beta, \\ \mathrm{Det}(\mathrm{H}) D_{xx} D_{yy} - D_{xy}^2 = \alpha\beta, \end{cases} \tag{5}$$

因此:

$$\mathrm{R} = \mathrm{Tr}(\mathrm{H})^2 / \mathrm{Det}(\mathrm{H}) = (r+1)^2 / r \tag{6}$$

其中 $r = \alpha/\beta$。由此可知当 R 越大，α、β 的比例越大，此时设定一阈值 r_{th}，若 R 大于 $(r_{\mathrm{th}}+1)^2/r_{\mathrm{th}}$ 表示此关键点位置不合需求因此可以去除。

一般来说，设定阈值 $r_{th} = 10$。

1.3.3 为每个关键点指定方向参数

在方位定向中，关键点以相邻像素的梯度方向分布作为指定方向参数，使关键点描述子能以根据此方向来表示并具备旋转不变性。

经高斯模糊处理后的影像 $L(x, y, \sigma)$，在 σ 尺寸下的梯度量 $m(x, y)$ 与方向 $\theta(x, y)$ 可由相邻之像素值计算:

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{7}$$

$$\theta(x,y) = \mathrm{atan}\,2(L(x,y+1) - L(x,y-1), L(x+1,y) - L(x-1,y)) \tag{8}$$

计算每个关键点与其相邻像素之梯度的量值与方向后，为其建立一个以 10 度为单位 36 条的直方图。每个相邻像素依据其量值大小与方向加入关键点的直方图中，最后直方图中最大值的方向即为此关键点的方向。若最大值与局部极大值的差在 20% 以内，则此判断此关键点含有多个方向，因此将会再额外建立一个位置、尺寸相同方向不同的关键点。但本次实验中我们没有使用这种处理手段。

1.3.4 关键点描述子的生成

SIFT 描述子的统计在相对物体坐标系以关键点为中心的 16Œ16 的领域内统计，先把之前计算的梯度方向由图像坐标系换算到物体坐标系，即

$$\theta'(x,y) = \theta(x,y) - \theta_0 \tag{9}$$

其中 θ' 是相对物体坐标系的梯度方向，θ 是相对图像坐标系的梯度方向，θ_0 是关键点的主方向。

物体坐标系 16×16 的邻域分成 4×4 个块，每个块 4×4 个像素。在每个块内按照求主方向的方式把 360 度分成 8 个 bins，统计梯度方向直方图，最终每个块可生成 8 维的直方图向量，每个关键点可生成 $4 \times 4 \times 8 = 128$ 维的 SIFT 描述子。

物体坐标系上的每一个整数点对应的图像坐标系可能不是整数，可采用最邻近插值，即图像坐标系上和它最接近的一个点:

$$\theta\left(x', y'\right) = \theta\left(\text{Round}\left(x'\right), \text{Round}\left(y'\right)\right) \tag{10}$$

或更精确地，可采用双线性插值:

$$\begin{aligned} \theta\left(x', y'\right) = \theta(x, y) * dx2 * dy2 \\ +\theta(x+1, y) * dx1 * dy2 \\ +\theta(x, y+1) * dx2 * dy1 \\ +\theta(x+1, y+1) * dx1 * dy1 \end{aligned} \tag{11}$$

双线性插值的意义在于，周围的四个点的值都对目标点有贡献，贡献大小与距离成正比。

最后对 128 维 SIFT 描述子 f_0 归一化得到最终的结果:

$$f = f_0 * \frac{1}{|f_0|} \tag{12}$$

两个 SIFT 描述子 f_1、f_2 之间的相似度可表示为:

$$s\left(f_1, f_2\right) = f_1 \cdot f_2 \tag{13}$$

2 实验过程

下面我将介绍我实现的各步的实验方法。篇幅和时间所限，我不打算详细展开每一步。最终的源代码在报告末附上。

我创建了类"MySIFT" 来实现我的 SIFT 算法。该类构造时接受原始图片和提取角点时的角点数量。

2.1 图片预处理

在 SIFT 正式步骤开始之前，我首先对图片进行了一些必须的预处理操作。

和之前一样，在进行特征提取之前，我们有必要将图片处理为灰度图，以将图像降为二维，滤过不必要的冗余信息。

```
def __graying(self):
    self.__img_float = np.dot(self.__img_ori[..., :3], [0.114, 0.587, 0.299])
    self.__img = cv2.convertScaleAbs(self.__img_float)
```

另外，图像的梯度幅值和方向是必不可少的，在接下来的实验过程中被大量使用。为此，我将其设为成员变量，用两个 Numpy 对象存储图像每个点的梯度幅值和梯度方向。

在计算时，我利用"filter2D" 得到两个方向的梯度矩阵，并通过 Numpy 运算直接进行后续操作，这样使算法效率得到很大提高。

```
def __get_grad(self):
    x_kernel = np.array([[0, 0, 0],
                         [-1, 0, 1],
                         [0, 0, 0]])

    y_kernel = np.array([[0, -1, 0],
                         [0, 0, 0],
                         [0, 1, 0]])

    grad_mag_x = cv2.filter2D(self.__img_float, -1, x_kernel)
    grad_mag_y = cv2.filter2D(self.__img_float, -1, y_kernel)

    self.__grad_mag = np.power((np.power(grad_mag_x, 2) + np.power(grad_mag_y, 2)), 0.5)
    self.__grad_dir = np.arctan2(grad_mag_y, grad_mag_x)
```

在计算角度时，我选用 Numpy 中的"arctan2" 方法。该方法能够输出 $-\pi$ 到 π 的角度，不必再进行进一步判断，避免了不必要的麻烦。

2.2 角点提取

该步骤是实验中的关键步骤，同时也是较为复杂的一部分。而由于时间受限，我没有自己实现这一部分的算法，而是借助 OpenCV 中的有关函数"goodFeaturesToTrack" 获取角点。

```
def __get_corners(self):
    self.__corners = cv2.goodFeaturesToTrack(self.__img,
        maxCorners=self.__corner_num,
        qualityLevel=0.01, minDistance=10, blockSize=3, k=0.04)

    self.__corners = self.__corners.astype(int)
```

值得注意的是，角点的坐标与 Numpy 中的表示有区别，在之后的过程中我会对其进行调整。

2.3 指定主方向

在提取了角点之后，我们需要对每个关键点进行描述子的计算，而计算是从主方向的确定开始的。

实现指定主方向的方法是" 投票"。顾名思义，对于每个关键点的 $m \times m$ 邻域内的所有点，该关键点都赋予其一定的投票权，共同决定这个关键点的方向指向。每个" 选民" 的权利因地位高低而不尽相同，拥有较大梯度幅值的点被赋予了更大的影响力。在这里，我将邻域大小设为 3×3

投票箱共有 36 个，分别代表 360 度的每个 10 度区间。通过线性的缩放，每个 $-\pi$ 到 π 的方向被缩放到 0 到 35 的一个单值上。

投票结束后，进行统计。通过 Numpy 方法"where" 找到投票最多的投票箱，并转换主方向为角度保存。

```
def __get_main_dir(self):
    self.__corner_dir = np.zeros((self.__y, self.__x))
```

```
    for corner in self.__corners:
        x = corner[0][0]
        y = corner[0][1]
        direct = [0] * 36

        for i in range(x - 1, x + 2):
            for j in range(y - 1, y + 2):
                if 0 <= i <= self.__x - 1 and 0 <= j <= self.__y - 1:
                    vote = int((self.__grad_dir[i, j] / np.pi + 1) * 18)
                    if vote == 36:
                        vote = 35
                    direct[vote] += self.__grad_mag[j, i]

        max_index = np.where(direct == np.max(direct))[0]    # find the max indies
        rank = max_index[0] + 1 # to 36 bins

        theta = rank * (np.pi / 18) - np.pi

        self.__corner_dir[y, x] = theta
```

值得注意的是，为了将边缘纳入 0 到 35 中的一个值，我对 vote 为 36 补充了判定条件，使其被归为 35 号投票箱。

2.4 描述子生成

为了实现描述子的生成，我首先定义了工具函数"get_BI" 来实现插值功能。

```
def __get_BI(self, x, y):
    x_left = int(x)
    x_right = x_left + 1
    y_low = int(y)
    y_high = y_low + 1

    dx1 = x - x_left
    dx2 = x_right - x
    dy1 = y - y_low
    dy2 = y_high - y

    return self.__grad_dir[y_low, x_left] * dx2 * dy2 + \
        self.__grad_dir[y_low, x_right] * dx1 * dy2 + \
        self.__grad_dir[y_high, x_left] * dx2 * dy1 + \
        self.__grad_dir[y_high, x_right] * dx1 * dy1
```

描述子的生成同样利用了投票的思想。

通过之前计算的主方向，我们能够得到物体坐标系中的 16×16 邻域，计算只需依赖 θ 角。对于每个 4×4 区域，我对每个小块进行插值，然后对插值结果投票。注意，我们需要统计的是物体坐标系中的方向，故要减去原来主方向。

该部分代码省略，可以在报告末查看。

2.5 匹配分析

该部分我分为匹配点个数和作出匹配图两部分来实现。(代码此处省略)

2.5.1 count_pair

函数"count_pair" 接受两幅图及其特征点，进行遍历匹配。通过计算两个归一化后的关键字间的欧氏距离，我们对某一匹配进行量化。对于没有查找过的点，如果该配对是好的匹配，则配对数加一。

2.5.2 draw_matches

作图函数"draw_matches" 与"count_pair" 类似。对于拥有匹配个数最多的两幅图，函数将对应的关键点进行连线。

两幅图的拼接首先建立一个空白图，然后通过复制两幅图的内容得到拼接图。匹配的连线则是用 OpenCV 中的"line" 方法实现的。

3 实验结果

根据我的上述实验，我能够从 5 张测试数据中选出匹配的图像，并作出匹配图。

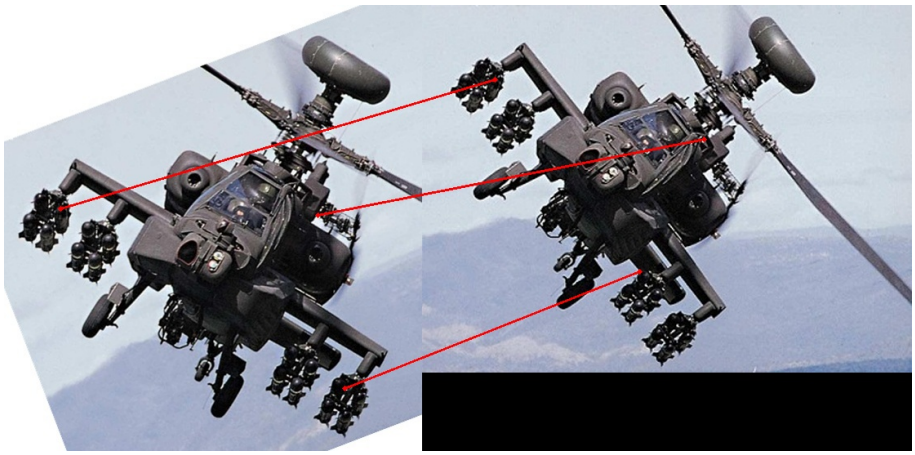

Fig2: 匹配效果图

4 结论分析

实验中我遇到了不少困难，在此我根据结果做一定的分析。

4.1 匹配点误差较大

首先，虽然我根据作业讲解实现了各步，但匹配效果并不十分理想。从匹配图中可以看出，匹配的关键点并不能完全匹配上。这可能是由于多方面原因引起的，但我认为图片间尺度的差异应该是主要原因。

对于比较的两幅图，我仅将两幅的尺寸放缩到同样尺度，这样实际上忽略了本身的尺度大小。一种更好的方法是通过选取不同的尺度对两幅图片进行比较，这样可以尽可能找到最正确的尺度。但由于时间有限，我没有实现这个方法。

5 源代码

```
import cv2
import numpy as np
from numpy import linalg as LA


class MySIFT(object):
    def __init__(self, img, corners):
        self.__img_ori = img
        self.__corner_num = corners

    def __graying(self):
        self.__img_float = np.dot(self.__img_ori[..., :3], [0.114, 0.587, 0.299])
        self.__img = cv2.convertScaleAbs(self.__img_float)

        self.__x = self.__img.shape[1]
        self.__y = self.__img.shape[0]

    def __get_grad(self):
        x_kernel = np.array([[0, 0, 0],
                             [-1, 0, 1],
                             [0, 0, 0]])

        y_kernel = np.array([[0, -1, 0],
                             [0, 0, 0],
                             [0, 1, 0]])

        grad_mag_x = cv2.filter2D(self.__img_float, -1, x_kernel)
        grad_mag_y = cv2.filter2D(self.__img_float, -1, y_kernel)

        self.__grad_mag = np.power((np.power(grad_mag_x, 2) + np.power(grad_mag_y, 2)), 0.5)
        self.__grad_dir = np.arctan2(grad_mag_y, grad_mag_x)

    # Parameters
    def __get_corners(self):
        self.__corners = cv2.goodFeaturesToTrack(self.__img,
                             maxCorners=self.__corner_num,
                             qualityLevel=0.01, minDistance=10, blockSize=3, k=0.04)

        self.__corners = self.__corners.astype(int)

    def __get_main_dir(self):
        self.__corner_dir = np.zeros((self.__y, self.__x))
        for corner in self.__corners:
            x = corner[0][0]
            y = corner[0][1]
            direct = [0] * 36

            for i in range(x - 1, x + 2):
                for j in range(y - 1, y + 2):
```

```
                if 0 <= i <= self.__x - 1 and 0 <= j <= self.__y - 1:
                    vote = int((self.__grad_dir[j, i] / np.pi + 1) * 18)
                    if vote == 36:
                        vote = 35
                    direct[vote] += self.__grad_mag[j, i]

        max_index = np.where(direct == np.max(direct))[0]    # find the max indies
        rank = max_index[0] + 1 # to 36 bins

        theta = rank * (np.pi / 18) - np.pi

        self.__corner_dir[y, x] = theta

def __get_BI(self, x, y):
    x_left = int(x)
    x_right = x_left + 1
    y_low = int(y)
    y_high = y_low + 1

    dx1 = x - x_left
    dx2 = x_right - x
    dy1 = y - y_low
    dy2 = y_high - y

    return self.__grad_dir[y_low, x_left] * dx2 * dy2 + \
           self.__grad_dir[y_low, x_right] * dx1 * dy2 + \
           self.__grad_dir[y_high, x_left] * dx2 * dy1 + \
           self.__grad_dir[y_high, x_right] * dx1 * dy1

def __get_descriptor(self):
    self.__result = {}

    for corner in self.__corners:
        x = corner[0][0]
        y = corner[0][1]
        grad_dir = self.__corner_dir[y, x]
        sin = np.sin(grad_dir)
        cos = np.cos(grad_dir)
        desc = np.zeros((4, 4, 8))

        for i in range(4):
            for j in range(4):
                Dx = 4 * i - 8  # [-8, -4, 0, 4]
                Dy = 4 * j - 8  # [-8, -4, 0, 4]

                # 块的一角
                main_x = x + Dx * cos
                main_y = y + Dy * sin

                for dx in range(4):
                    for dy in range(4):
                        # 元素的一角
```

```
                            minor_x = main_x + dx * cos
                            minor_y = main_y + dy * sin

                            if 0 <= minor_x < self.__x - 1 and 0 <= minor_y < self.__y - 1:
                                theta = self.__get_BI(minor_x, minor_y) - self.__corner_dir[y,
                                                                                            x]
                                if theta > np.pi:
                                    theta -= np.pi
                                elif theta < -np.pi:
                                    theta += np.pi

                                vote = int((theta / np.pi + 1) * 4)
                                if vote == 8:
                                    vote = 7
                                desc[i, j, vote] += 1

            desc = np.reshape(desc, (1, 128))    # 可不用
            self.__result[(x, y)] = desc

    def show_corner(self):
        corner_img = self.__img_ori * 1

        point_size = 1
        point_color = (0, 0, 255)  # BGR
        thickness = 4  # 可以为 0 、4、8

        for corner in self.__corners:
            cv2.circle(corner_img, tuple(corner[0]), point_size, point_color, thickness)

        cv2.imshow("corners", corner_img)
        k = cv2.waitKey(0)

    def test(self):
        self.__graying()
        # self.__get_corners()
        self.__get_grad()
        self.__get_corners()
        self.__get_main_dir()
        self.__get_descriptor()

        return self.__result


def count_pair(img1, kp1, img2, kp2):
    visited1 = np.zeros((img1.shape[0], img1.shape[1]))
    visited2 = np.zeros((img2.shape[0], img2.shape[1]))
    num = 0

    for corner1, desc1 in kp1.items():
        check = 0

        for corner2, desc2 in kp2.items():
```

```
            if visited1[corner1[1], corner1[0]] or visited2[corner2[1], corner2[0]]:
                continue

            desc1 /= LA.norm(desc1)
            desc2 /= LA.norm(desc2)
            res = np.linalg.norm(desc1 - desc2)

            if res < 0.8:
                check = 1
                visited1[corner1[1], corner1[0]] = 1
                visited2[corner2[1], corner2[0]] = 1

            num += check

    return num


def draw_matches(img1, kp1, img2, kp2):
    visited1 = np.zeros((img1.shape[0], img1.shape[1]))
    visited2 = np.zeros((img2.shape[0], img2.shape[1]))
    r1 = img1.shape[0]
    c1 = img1.shape[1]
    r2 = img2.shape[0]
    c2 = img2.shape[1]

    out = np.zeros((max([r1, r2]), c1 + c2, 3), dtype='uint8')

    out[:r1, :c1, :] = np.dstack([img1])
    out[:r2, c1:c1 + c2, :] = np.dstack([img2])

    point_size = 1
    point_color = (0, 0, 255)  # BGR
    thickness = 4  # 可以为 0 、4、8

    for corner1, desc1 in kp1.items():
        for corner2, desc2 in kp2.items():
            if visited1[corner1[1], corner1[0]] or visited2[corner2[1], corner2[0]]:
                continue
            desc1 /= LA.norm(desc1)
            desc2 /= LA.norm(desc2)
            res = np.linalg.norm(desc1 - desc2)

            if res < 0.7:
                x1, y1 = corner1
                x2, y2 = corner2
                cv2.circle(out, corner1, point_size, point_color, thickness)
                cv2.circle(out, (int(x2) + c1, int(y2)), point_size, point_color, thickness)
                cv2.line(out, corner1, (int(x2) + c1, int(y2)), point_color, 2)

                visited1[y1, x1] = 1
                visited2[y2, x2] = 1
```

```
    cv2.imshow("test", out)
    cv2.imwrite("1.jpg", out)

    k = cv2.waitKey(0)
    if k == 'q':
        cv2.destroyAllWindows()


def main():
    img1 = cv2.imread("target.jpg")
    x1, y1 = (img1.shape[0], img1.shape[1])
    img1 = cv2.resize(img1, (512, int(512 * x1 / y1)), interpolation=cv2.INTER_CUBIC)
    sift1 = MySIFT(img1, corners=20)
    result1 = sift1.test()

    pair_num = [0] * 6
    for i in range(1, 6):
        img2 = cv2.imread("dataset/{0}.jpg".format(i))
        x2, y2 = (img2.shape[0], img2.shape[1])
        img2 = cv2.resize(img2, (512, int(512 * x2 / y2)), interpolation=cv2.INTER_CUBIC)

        sift2 = MySIFT(img2, corners=20)
        result2 = sift2.test()
        pair_num[i] = count_pair(img1, result1, img2, result2)

    max_index = np.where(pair_num == np.max(pair_num))[0]
    max_index = max_index[0]

    print("pic {0} is the best pair!".format(max_index))

    img2 = cv2.imread("dataset/{0}.jpg".format(max_index))
    x2, y2 = (img2.shape[0], img2.shape[1])
    img2 = cv2.resize(img2, (512, int(512 * x2 / y2)), interpolation=cv2.INTER_CUBIC)
    sift2 = MySIFT(img2, corners=20)
    result2 = sift2.test()

    draw_matches(img1, result1, img2, result2)

if __name__ == '__main__':
    main()
```