1. 使用方法

在命令行中输入python seam\_carving.py -i 输入图片路径 -o 输入图片路径。

例如：python seam\_carving.py -i ./pict.jpg -o ./pict\_resize2.jpg

1. 寻找能量最小路径算法

def cumulative\_map\_forward(self, energy\_map):

        m, n = energy\_map.shape

        dp = np.copy(energy\_map)

        # 记录从哪来的，-1左上，0上，1右上

        path = np.zeros((m, n))

        # 从上到下

        for i in range(1, m):

            for j in range(n):

                # 第一列没有左上

                if j == 0:

                    # 取上

                    if energy\_map[i - 1, j] < energy\_map[i - 1, j + 1]:

                        path[i, j] = 0

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j]

                    # 取右上

                    else:

                        path[i, j] = 1

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j + 1]

                # 最后一列没有右上

                elif j == n - 1:

                    # 取左上

                    if energy\_map[i - 1, j - 1] < energy\_map[i - 1, j]:

                        path[i, j] = -1

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j - 1]

                    # 取上

                    else:

                        path[i, j] = 0

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j]

                # 中间的有三个来源方向

                else:

                    # 取左上

                    if energy\_map[i - 1, j - 1] < min(energy\_map[i - 1, j], energy\_map[i - 1, j + 1]):

                        path[i, j] = -1

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j - 1]

                    # 取上

                    elif energy\_map[i - 1, j] < energy\_map[i - 1, j + 1]:

                        path[i, j] = 0

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j]

                    # 取右上

                    else:

                        path[i, j] = 1

                        dp[i, j] = energy\_map[i, j] + energy\_map[i - 1, j + 1]

        return dp, path

    # 通过回溯路径数组找seam位于每行的第几列

    # 输入：动态规划数组、回溯数组

    # 输出：seam位于每行的列号数组

    def find\_seam(self, dp, path):

        m, n = dp.shape

        seam\_idx = np.zeros((m, ), dtype=np.uint32)

        # 终点最小值对应的索引

        seam\_idx[-1] = np.argmin(dp[-1])

        # 逆向寻找最小的路径

        for i in range(m - 2, -1, -1):

            j = seam\_idx[i + 1]

            seam\_idx[i] = j + path[i, j]

        return seam\_idx

寻找能量最小路径算法分为两个步骤，分别为正向遍历填写动态规划状态表以及记录路径、以及逆向遍历求出最终路径。

正向遍历使用动态规划算法，建立和能量图相同大小的二维数组dp和记录路径的二维数组path，其中dp[i][j]表示从第一行到第j行的能量最小的路径能量和，path[i][j]表示除了第一列以外的位于(i, j)位置的能量点加合从何而来，其中-1表示左上，0表示上，1表示右上。该填表的过程中对一任意一个位置需要比较上一行至多三个数的大小，因此时间复杂度为O(n2)。

逆向遍历利用path数组，从最后一行的最小数字出发向上遍历，直到找到所经历的seam位于每行的列的索引。时间复杂度为O(n)

利用到和能量图相同大小的dp和path数组，因此空间复杂度为O(n2)。

因此该算法的时间复杂O(n2)，空间复杂度为O(n2)。