Seam Carving

本项目实现Seam Carving算法的图像缩放功能。包含两个版本：使用 for 循环实现和使用 numpy 向量化加速实现的寻找最短路径算法，其中后者的执行速度远远快于前者。

文件结构

```
├── README.md
├── output
│   ├── out_image1.png
│   ├── out_image2.png
│   └── out_image3.png
├── seam_carving.py
├── seam_carving_faster.py
└── src
    ├── image1.png
    ├── image2.png
    └── image3.png
```

项目的主要代码为 seam_carving.py 和 seam_carving_faster.py。seam_carving.py 是使用 for 循环实现寻找最短路径的版本，seam_carving_faster.py 是使用 numpy 向量化加速实现寻找最短路径的版本，推荐使用后者。

文件夹 src 存放输入图片，文件夹 output 存储经过缩放处理的图片。请注意调整代码中的文件路径！！！

运行指南

本项目基于 Python 编程语言，用到的外部代码库主要包括 numpy 和 cv2。程序运行使用的 Python 版本为 3.8.5。

环境配置

```
# 安装外部代码库
pip install -r requirements.txt
```

运行测试

可以直接运行以下指令进行测试：

```
cd code1
python seam_carving.py
python seam_carving_faster.py
```

寻找能量最小路径算法

针对寻找能量最小路径算法的时间复杂度和空间复杂度分析，选择 for 循环版本进行分析。

```
    def FindPerSeam(self):
        """
        从 out_image 中找到一个 seam
        """

        energy_map = self.CalculateEnergy()  # 计算能量图
        rows, cols = energy_map.shape

        # 最小能量值 = 该像素的能量值 + 上一行相邻像素的最小能量值
        M = energy_map.copy().astype(np.int32)  # 存每个像素的最小能量值

        # 从第二行开始，对应行号1
        for i in range(1, rows):
            for j in range(0, cols):
                if j == 0:  # 第一列
                    M[i, j] += min(M[i-1, j], M[i-1, j+1])
                elif j == cols - 1:  # 最后一列
                    M[i, j] += min(M[i-1, j-1], M[i-1, j])
                else:
                    M[i, j] += min(M[i-1, j-1], M[i-1, j], M[i-1,
j+1])

        # seam 保存我们需要删除的像素点坐标
        seam = np.zeros((rows, 2), dtype=np.int32)

        # 寻找最后一行 M[cols-1] 中值最小的一列，从下往上找
        min_j = np.argmin(M[rows-1])
        seam[rows-1, 1] = min_j  # 把最后一行的最小值的坐标加入seam

        for i in range(rows-1, -1, -1):  # 从最后一行找到第一行
            if (min_j == 0):
                min_j = np.argmin(M[i-1, min_j:min_j+2])
            elif (min_j == cols - 1):
                min_j += np.argmin(M[i-1, min_j-1:min_j+1]) - 1
            else:
                min_j += np.argmin(M[i-1, min_j-1:min_j+2]) - 1
            seam[i] = [i, min_j]

        return seam
```

时间复杂度

```
# 从第二行开始，对应行号1
for i in range(1, rows):
    for j in range(0, cols):
        if j == 0:  # 第一列
            M[i, j] += min(M[i-1, j], M[i-1, j+1])
        elif j == cols - 1:  # 最后一列
            M[i, j] += min(M[i-1, j-1], M[i-1, j])
        else:
            M[i, j] += min(M[i-1, j-1], M[i-1, j], M[i-1, j+1])
```

创建一个二维数据变量 M 来存储每个像素的最小能量值，通过两层 for 循环对 M[i][j] 进行复制，这部分时间复杂度为：$\Theta(rows * cols)$，其中 rows 为图片行数，cols 为图片列数。

下面一个单层 for 循环的时间复杂度为： $\Theta(rows)$，因此本函数总时间复杂度为：$\Theta(rows * cols)$。

上面时间复杂度分析是针对移除一个 seam 来说的，对于一张尺寸为 ROWS * COLS 的图片来说，如果要将其缩放至 ROWS * COLS/2，此时总时间复杂度大约为：$\Theta(ROWS * COLS^2)$

空间复杂度

创建一个二维数据变量 M 来存储每个像素的最小能量值，因此空间复杂度为：$\Theta(ROWS * COLS)$。

分析

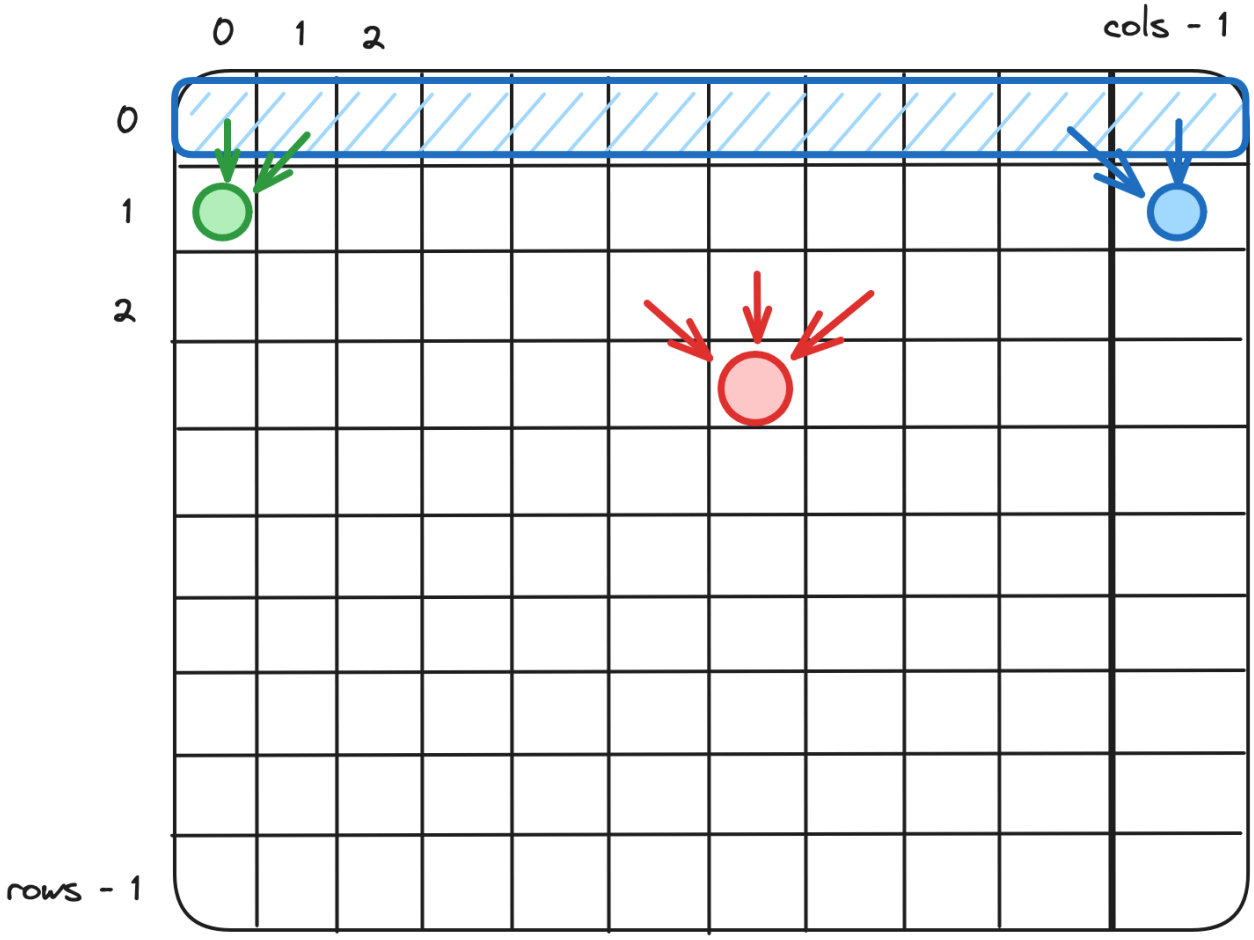
Seam Carving算法是一种用于图像缩放的算法，它可以在不改变图像比例的情况下，自动删除或插入像素，从而实现图像的缩放。

其基本思路如下：

1. 计算能量图：首先，需要计算出图像中每个像素的能量值。能量值可以根据像素的颜、梯度等特征来计算，用于衡量像素的重要性。
2. 找到最小能量路径：接下来，需要找到一条从图像顶部到底部的路径，使得路径上的像素能量之和最小。这条路径称为最小能量路径。
3. 删除最小能量路径：将最小能量路径上的像素删除，即可将图像宽度缩小1个像素。如果需要将图像高度缩小，则需要找到从左侧到右侧的最小能量路径，并删除路径上的像素。

重复步骤2和3：重复执行步骤2和3，直到达到所需的图像大小。

蓝色虚线矩阵框的地方：初始值等于能量图对应位置的值



时间对比

在写完算法后，我选择一张影视飓风官网[https://www.ysjf.com/index]图片进行测试，没有考虑到图片的画质和尺寸大小，因此算法跑得非常慢，经过网上查阅资料后，发现使用 numpy 将 for 循环向量化可以大大加速执行时间。具体效果对比图如下：

	image1.png	image2.png	image3.png
for 循环	约 3 x 960 s	7.172824s	13.646205s
numpy 向量	196.439090s	1.193867s	1.846587s

图：使用 numpy 向量化优化先后执行时间对比