Seam Carving Project

در این پروژه نحوه تقسیم وظایف بصورت زیر هست :

نام اعضا – شماره دانشجویی	وظایف
محسن احمدی – ۴۰۰۰۳۴۷۳	پیاده سازی تابع انرژی مپ و ایجاد رابط گرافیکی (GUI)
امیر مهدی اسلامی – ۴۰۱۰۰۴۳۳	پیاده سازی توابع: پیدا کردن Seam مینیمم و حذف آن، انجام فرایند Seam Carving در راستای افقی و عمودی

توجه :

۱. **در هنگام انتخاب فولدر حتما باید سه تصویر داخل فولدر باشد. تصویر اصلی، انرژی مپ عمق و برجستگی. فرمت نام گذاری تصاویر عمق حتما باید بصورت .Name_DMap و تصاویر برجستگی بصورت .Name_SMap باشد تا اپلیکیشن تشخیص دهد تمایز این تصاویر را برای انجام فرایند.**

۲. **برای نصب کتابخانه های در همان دایرکتوری اصلی کد، یک فایل به نام req.txt وجود دارد کافی است دستور زیر را اجرا کنید:**

Pip install -r req.txt

۳. **به دلیل حجم زیاد فایل های ویدیویی یک فایل Video.txt در کنار فایل پروژه قرار گرفته است و لینک iutbox ویدیو های توضیحات پروژه در آن جا قرار گرفته است.**

۴. **خروجی های بدست آمده در صورت نیاز برای ذخیره سازی حتما در یک فولدر جداگانه ذخیره کنید.**

۵. **تمام تصاویر داده شده با ۵۰ درصد کاهش در کنار فایل های پروژه قرار دارد.**

توضیحات تابع انرژی مپ :

از تصاویر Saliency, Depth که از قبل بصورت آماده به همراه محاسبه مشتق اول تصویر که با استفاده از کرنل Soble و مشتق دوم تصویر که توسط کرنل Laplacian بدست آماده این فرایند انجام شده. تمام این موارد بصورت جمع ضریب دار باهم ترکیب شده اند و انرژی مپ نهایی بوجود آماده هست.

تصاویر Saliency, Depth ابتدا تابع نرمالایز شدن و بازه آن ها بین ۰ و ۲۵۵ قرار میگیرد.

```
# Normalize depth map
depth_map = cv2.normalize(depth_map, None, 0, 255, cv2.NORM_MINMAX).astype(np.float32)

# Normalize saliency map
saliency_map = cv2.normalize(saliency_map, None, 0, 255, cv2.NORM_MINMAX).astype(np.float32)
```

بعد از این فرایند تصویر اصلی خوانده شده از فضای رنگی RGB به یک تصویر grayscale تبدیل می‌شود تا در مراحل بعد مشتق اول و دوم توسط این تصویر grayscale بدست آید.

```
# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY).astype(np.float32)
```

در این مرحله مشتق اول و دوم محاسبه می شوند و خروجی آن ها نرمالایز می شود.

```
# Compute gradients
grad_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
grad_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
gradient_map = cv2.magnitude(grad_x, grad_y)
gradient_map = cv2.normalize(gradient_map, None, 0, 255, cv2.NORM_MINMAX)

# Compute Laplacian
laplacian = cv2.Laplacian(gray, cv2.CV_32F)
laplacian = cv2.normalize(laplacian, None, 0, 255, cv2.NORM_MINMAX)
```

در نهایت این خروجی ها بصورت جمع ضریب دار با هم ترکیب می شوند.

```
# Calculate energy map
energy_map = (alpha * depth_map + beta * saliency_map + gamma * gradient_map + rho * laplacian)
```

این ضرایب با توجه به میزان اهمیت هر تصویر تعیین شده و با آزمایش. به این صورت که میزان تاثیر تصویر عمق به دلیل اینکه انرژی مپ بهتری داشته و خروجی بهتر تولید میکند ضریب آن مقدار بیشتری دارد و سایر ضرایب باتوجه به اهمیتی که داشتند مقادیر کمتری دارند.

```
alpha: float = 2.0, beta: float = 0.58, gamma: float = 0.7, rho: float = 0.8
```

توجه: این ضرایب بصورت تجربی و با تست بدست آمده.

توضیح تابع crop_column :

این تابع سه تا ورودی میگیرد

img : آرایه‌ای که تصویر اصلی را نشان می‌دهد.

energy_map : نقشه انرژی تصویر که نشان‌دهنده اهمیت هر پیکسل است.

scale_c : ضریب مقیاس‌بندی برای کاهش تعداد ستون‌ها.

تعداد کل ستون هایی که باید حذف شوند را پیدا می کنیم

```
total_steps = math.ceil(scale_c * cols)
```

یک حلقه ی for برای حذف ستون ها میسازیم و در هر مرحله تابع remove_columns را صدا میزنیم که وظیفش حذف ستون ها براساس energy_map است

```
for step in range(total_steps):
    img, energy_map = self.remove_column(img, energy_map)
```

توضیح تابع crop_row :

اگر بخواهیم به صورت سطری تصویرمان را کوچک کنیم از این تابع استفاده می کنیم

ابتدا انرژی مپ و تصویر اصلیمان را ۹۰ درجه دوران می دهیم

```
img = np.rot90(img, 1, (0, 1))
energy_map = np.rot90(energy_map, 1, (0, 1))
```

و سپس تابع crop_columns را براش صدا میزنیم در حقیقت با دوران ۹۰ درجه دادن به تصویر و انرژی مپمان کاری کردیم که از ان ها به صورت افقی بتوانیم استفاده بکنیم و تصویرمان را کوچک بکنیم

توضیح تابع remove_colmns :

این تابع یک ستون از تصویر را حذف می‌کند . ورودی های تصویر به صورت زیر است:

img : آرایه‌ای که تصویر را نشان می‌دهد.

energy_map : نقشه انرژی تصویر.

ابعاد تصویر را محاسبه می‌کند. محور مورد نظر برای حذف ستون را تعیین می‌کند

```
rows, cols, _ = img.shape
which_axis = self.axis_var.get()
```

تابع find_min_seamرا صدا می‌زند تا کم‌اهمیت ترین درز تصویر را پیدا کند.

```
M, backtrack = self.find_min_seam(energy_map)
```

یک ماسک بولی با ابعاد تصویر ایجاد می‌کند که در ابتدا همه عناصر آن `True` هستند .این ماسک برای مشخص کردن پیکسل‌هایی که باید حفظ شوند استفاده می‌شود.

```
mask = np.ones((rows, cols), dtype=bool)
```

ستون پایینی با کمترین انرژی را در ماتریس `M` پیدا می‌کند و اندیس آن را در `j` ذخیره می‌کند .این ستون شروع درز است.

```
j = np.argmin(M[-1])
```

یک حلقه برای پیدا کردن پیکسل‌های درز از پایین به بالا اجرا می‌شود

اگر `j` کمتر از تعداد ستون‌ها باشد، پیکسل مربوطه را به عنوان بخشی از درز مشخص می‌کند و به لیست `seam` اضافه می‌کند.

`J` را با استفاده از ماتریس `backtrack` به ستون قبلی در درز تغییر می‌دهد.

```
for i in reversed(range(rows)):
    if j < cols:
        mask[i, j] = False
        seam.append((i, j))
        j = backtrack[i, j]
```

ماسک را به یک آرایه سه بعدی تبدیل می‌کند تا با شکل تصویر مطابقت داشته باشد.

```
mask = np.stack([mask] * 3, axis=2)
```

پیکسل‌های درز را در کپی تصویر به رنگ قرمز تغییر می‌دهد و در کپی نقشه انرژی به صفر تنظیم می‌کند.

```
for (i, j) in seam:
    if j < temp_img.shape[1]:
        temp_img[i, j] = [255, 0, 0]
        temp_energy_map[i, j] = 0  # Highlight the seam in the energy map
```

تصویر و نقشه انرژی را با استفاده از ماسک تغییر شکل می‌دهد تا ستون حذف شود.

```
energy_map = energy_map[mask[:, :, 0]].reshape((rows, cols - 1))
img = img[mask].reshape((rows, cols - 1, 3))
```

اگر محور حذف سطر باشد، تصویر، نقشه انرژی و کپی‌های آن‌ها را ۹۰ درجه می‌چرخاند.

```
if which_axis == 'r':
    # temp_img = np.rot90(temp_img, 1, (0, 1))
    img = np.rot90(img, 1, (0, 1))
    # temp_energy_map = np.rot90(temp_energy_map, 1, (0, 1))
    energy_map = np.rot90(energy_map, 1, (0, 1))
```

توضیح تابع find_min_seam

این تابع برای پیدا کردن کم‌هزینه‌ترین مسیر در یک نقشه انرژی استفاده می‌شود .این مسیر در الگوریتم seam_carving برای حذف پیکسل‌ها استفاده می‌شود.

```
def find_min_seam(self, energy_map: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
```

یک ورودی دارد که انرژی مپ ما است.

```
rows, cols = energy_map.shape
M = energy_map.copy()
backtrack = np.zeros_like(M, dtype=int)
```

تعداد سطر و ستون هاو مشخص میکند و یک کپی از انرژی مپ میگیرد و یک آرایه صفر با همان شکل M ایجاد می‌کند و آن را در متغیر backtrack قرار می‌دهد .این ماتریس برای ذخیره اطلاعات بازگشتی در حین محاسبه مسیر استفاده می‌شود.

```
for i in range(1, rows):
    for j in range(0, cols):
        if j == 0:
            idx = np.argmin(M[i - 1, j:j + 2])
            backtrack[i, j] = idx + j
            min_energy = M[i - 1, idx + j]
        else:
            idx = np.argmin(M[i - 1, j - 1:j + 2])
            backtrack[i, j] = idx + j - 1
            min_energy = M[i - 1, idx + j - 1]
        M[i, j] += min_energy
return M, backtrack
```

این حلقه دو وظیفه دارد

محاسبه هزینه برای هر پیکسل در ماتریس M

تعیین مسیر بهینه (کم‌هزینه) از هر پیکسل به سطر بالا با استفاده از ماتریس backtrack

که این اعداد را برای هر پیکسل از محاسبه ی سه پیکسل بالاییش بدست می أورد