计算机视觉实验教程 - 图像重定向算法

陈俊周 中山大学智能工程学院

1. 图像重定向算法

1.1 颜色对比度显著性

```
    def color_contrast_saliencyimage):
        """基于颜色对比度的显著性检测"""
        # 转换到LAB色彩空间
        lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
        # 计算图像的平均颜色
        mean_lab = np.mean(lab, axis=(0, 1))
        # 计算每个像素与平均颜色的欧氏距离
        saliency = np.sqrt(np.sum((lab - mean_lab) ** 2, axis=2))
        # 归一化到[0, 255]
        saliency = (saliency - saliency.min()) / (saliency.max() - saliency.min())
        saliency = (saliency * 255).astype(np.uint8)

        return saliency
```

1.2 中心先验显著性

```
    def center_prior_saliency(image):
        """基于中心先验的显著性检测"""
        h, w = image.shape[:2]
        # 创建中心先验图
        center_x, center_y = w // 2, h // 2
        Y, X = np.ogrid[:h, :w]
        # 计算到中心的距离
        dist_from_center = np.sqrt((X - center_x) ** 2 + (Y - center_y) ** 2)
        max_dist = np.sqrt(center_x ** 2 + center_y ** 2)
        # 使用高斯函数建模中心先验
        center_prior = np.exp(-(dist_from_center ** 2) / (2 * (max_dist / 2) ** 2))
        center_prior = (center_prior * 255).astype(np.uint8)

        return center_prior
```

1.3 谱残差显著性

```
    def spectral_residual_saliency(image):
        """谱残差显著性检测"""
        # 转换为灰度图
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        gray = gray.astype(np.float32) / 255.0
        # 傅里叶变换
```

```
        fft = np.fft.fft2(gray)
        log_amplitude = np.log(np.abs(fft) + 1e-7)
        phase = np.angle(fft)
        # 计算谱残差
        spectral_residual = log_amplitude - gaussian_filter(log_amplitude, sigma=3)
        # 重建图像
        combined = np.exp(spectral_residual + 1j * phase)
        saliency = np.abs(np.fft.ifft2(combined))
        # 后处理
        saliency = gaussian_filter(saliency ** 2, sigma=8)
        saliency = (saliency - saliency.min()) / (saliency.max() - saliency.min())
        saliency = (saliency * 255).astype(np.uint8)
        return saliency
```

1.3 Seam Carving 算法

```
import numpy as np
import cv2
from scipy.ndimage import convolve
import matplotlib.pyplot as plt


class SeamCarving:
    """Seam Carving图像重定位算法"""

    def __init__(self, image, saliency_map=None):
        """
        初始化Seam Carving

        Args:
            image: 输入图像
            saliency_map: 显著性图（可选）
        """
        self.original_image = image.copy()
        self.image = image.copy()
        self.saliency_map = saliency_map

    def calculate_energy(self, use_saliency=True):
        """
        计算能量图

        Args:
            use_saliency: 是否使用显著性图增强能量
        """
        # 转换为灰度图
        gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
        # Sobel梯度
        dx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
        dy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
```

```
        # 梯度幅值作为基础能量
        energy = np.abs(dx) + np.abs(dy)
        # 如果有显著性图，将其融合到能量图中
        if use_saliency and self.saliency_map is not None:
            # 调整显著性图大小以匹配当前图像
            resized_saliency = cv2.resize(self.saliency_map,
                                          (self.image.shape[1],
self.image.shape[0]))
            # 融合显著性信息（显著区域能量更高）
            saliency_weight = 1000  # 权重系数
            energy = energy + saliency_weight * (resized_saliency / 255.0)

        return energy

    def find_vertical_seam(self, energy):
        """
        使用动态规划找到最小能量垂直缝

        Args:
            energy: 能量图

        Returns:
            seam: 垂直缝的列索引数组
        """
        h, w = energy.shape
        # 动态规划表
        dp = energy.copy()
        # 从第二行开始，计算累积最小能量
        for i in range(1, h):
            for j in range(w):
                # 检查上一行的三个可能位置
                if j == 0:
                    dp[i, j] += min(dp[i - 1, j], dp[i - 1, j + 1])
                elif j == w - 1:
                    dp[i, j] += min(dp[i - 1, j - 1], dp[i - 1, j])
                else:
                    dp[i, j] += min(dp[i - 1, j - 1], dp[i - 1, j], dp[i - 1, j +
1])
        # 回溯找到最小能量缝
        seam = np.zeros(h, dtype=np.int32)
        seam[-1] = np.argmin(dp[-1])
        for i in range(h - 2, -1, -1):
            j = seam[i + 1]
            if j == 0:
                seam[i] = j + np.argmin(dp[i, j:j + 2])
            elif j == w - 1:
                seam[i] = j - 1 + np.argmin(dp[i, j - 1:j + 1])
            else:
                seam[i] = j - 1 + np.argmin(dp[i, j - 1:j + 2])
        return seam

    def find_horizontal_seam(self, energy):
```

```
        """找到最小能量水平缝"""
        # 转置能量图，使用垂直缝算法，然后转换回来
        energy_T = energy.T
        seam = self.find_vertical_seam(energy_T)
        return seam

    def remove_vertical_seam(self, seam):
        """移除垂直缝"""
        h, w, c = self.image.shape
        output = np.zeros((h, w - 1, c), dtype=self.image.dtype)

        for i in range(h):
            j = seam[i]
            output[i, :, :] = np.delete(self.image[i, :, :], j, axis=0)

        self.image = output

        # 同时更新显著性图
        if self.saliency_map is not None:
            h_s, w_s = self.saliency_map.shape
            output_saliency = np.zeros((h_s, w_s - 1),
dtype=self.saliency_map.dtype)

            # 调整seam以匹配显著性图的尺寸
            seam_scaled = (seam * w_s / w).astype(int)

            for i in range(h_s):
                j = seam_scaled[min(i, len(seam_scaled) - 1)]
                output_saliency[i, :] = np.delete(self.saliency_map[i, :], j,
axis=0)

            self.saliency_map = output_saliency

    def remove_horizontal_seam(self, seam):
        """移除水平缝"""
        self.image = np.transpose(self.image, (1, 0, 2))
        self.remove_vertical_seam(seam)
        self.image = np.transpose(self.image, (1, 0, 2))

        if self.saliency_map is not None:
            self.saliency_map = self.saliency_map.T
            # 这里简化处理，实际应该也要调整seam
            self.saliency_map = self.saliency_map.T

    def resize(self, new_width=None, new_height=None, use_saliency=True):
        """
        调整图像大小

        Args:
            new_width: 目标宽度
            new_height: 目标高度
            use_saliency: 是否使用显著性引导
```

```
        """
        if new_width is not None:
            delta_w = self.image.shape[1] - new_width

            if delta_w > 0:  # 缩小宽度
                for _ in range(delta_w):
                    energy = self.calculate_energy(use_saliency)
                    seam = self.find_vertical_seam(energy)
                    self.remove_vertical_seam(seam)

                    if _ % 10 == 0:
                        print(f"已移除 {_ + 1}/{delta_w} 条垂直缝")

        if new_height is not None:
            delta_h = self.image.shape[0] - new_height

            if delta_h > 0:  # 缩小高度
                for _ in range(delta_h):
                    energy = self.calculate_energy(use_saliency)
                    seam = self.find_horizontal_seam(energy)
                    self.remove_horizontal_seam(seam)

                    if _ % 10 == 0:
                        print(f"已移除 {_ + 1}/{delta_h} 条水平缝")

        return self.image

    def visualize_seam(self, seam, direction='vertical'):
        """可视化缝的位置"""
        vis_image = self.image.copy()

        if direction == 'vertical':
            for i in range(len(seam)):
                vis_image[i, seam[i]] = [0, 0, 255]  # 红色标记
        else:  # horizontal
            for i in range(len(seam)):
                vis_image[seam[i], i] = [0, 0, 255]

        return vis_image


# 测试代码
def test_seam_carving():
    """测试Seam Carving算法"""
    # 创建测试图像
    image_path = 'sysu-sz.jpg'
    test_image = cv2.imread(image_path)

    if test_image is None:
        raise FileNotFoundError(f"无法读取图像: {image_path}")

    # 创建简单的显著性图（标记重要区域）
```

```python
    saliency = cv2.imread('saliency_map.png', cv2.IMREAD_GRAYSCALE)

    # 初始化Seam Carving
    sc = SeamCarving(test_image, saliency)

    # 计算并可视化第一条缝
    energy = sc.calculate_energy(use_saliency=True)
    seam = sc.find_vertical_seam(energy)
    seam_vis = sc.visualize_seam(seam)

    # 调整图像大小
    print("原始大小:", test_image.shape[:2])
    resized = sc.resize(new_width=400, use_saliency=True)
    print("调整后大小:", resized.shape[:2])

    # 可视化结果
    fig, axes = plt.subplots(2, 3, figsize=(15, 10))

    axes[0, 0].imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
    axes[0, 0].set_title('Original Image')
    axes[0, 0].axis('off')

    axes[0, 1].imshow(saliency, cmap='hot')
    axes[0, 1].set_title('Saliency Map')
    axes[0, 1].axis('off')

    axes[0, 2].imshow(energy, cmap='hot')
    axes[0, 2].set_title('Energy Map')
    axes[0, 2].axis('off')

    axes[1, 0].imshow(cv2.cvtColor(seam_vis, cv2.COLOR_BGR2RGB))
    axes[1, 0].set_title('First Seam')
    axes[1, 0].axis('off')

    axes[1, 1].imshow(cv2.cvtColor(resized, cv2.COLOR_BGR2RGB))
    axes[1, 1].set_title(f'Resized Image ({resized.shape[1]}x{resized.shape[0]})')
    axes[1, 1].axis('off')

    # 对比不使用显著性的结果
    sc_no_sal = SeamCarving(test_image, None)
    resized_no_sal = sc_no_sal.resize(new_width=400, use_saliency=False)

    axes[1, 2].imshow(cv2.cvtColor(resized_no_sal, cv2.COLOR_BGR2RGB))
    axes[1, 2].set_title('No Saliency Map Resized Image')
    axes[1, 2].axis('off')

    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    test_seam_carving()
```

2. 质量评估

2.1 PSNR

```python
import numpy as np
import math

def calculate_psnr(original, compressed):
    """
    计算PSNR（峰值信噪比）

    Args:
        original: 原始图像
        compressed: 处理后图像

    Returns:
        PSNR值（单位：dB）
    """
    # 确保图像尺寸相同
    assert original.shape == compressed.shape

    # 转换为浮点数避免溢出
    original = original.astype(np.float64)
    compressed = compressed.astype(np.float64)

    # Step 1: 计算MSE
    mse = np.mean((original - compressed) ** 2)

    # 如果MSE为0，说明两张图完全一样
    if mse == 0:
        return float('inf')

    # Step 2: 计算PSNR
    max_pixel = 255.0  # 8位图像的最大值
    psnr = 10 * math.log10(max_pixel ** 2 / mse)

    return psnr

# 使用示例
psnr_value = calculate_psnr(original_img, processed_img)
print(f"PSNR: {psnr_value:.2f} dB")
```

2.2 SSIM

```
```

```python
import numpy as np
from scipy.ndimage import gaussian_filter

def calculate_ssim(img1, img2, window_size=11):
    """
    计算SSIM（结构相似性指数）

    Args:
        img1, img2: 要比较的两张图像
        window_size: 滑动窗口大小

    Returns:
        SSIM值（0-1之间）
    """
    # 转换为浮点数
    img1 = img1.astype(np.float64)
    img2 = img2.astype(np.float64)

    # 常数设置（避免除零）
    K1 = 0.01
    K2 = 0.03
    L = 255  # 像素值范围
    C1 = (K1 * L) ** 2
    C2 = (K2 * L) ** 2

    # 使用高斯窗口计算局部统计
    sigma = 1.5

    # 计算均值
    mu1 = gaussian_filter(img1, sigma)
    mu2 = gaussian_filter(img2, sigma)

    # 计算方差和协方差
    mu1_sq = mu1 ** 2
    mu2_sq = mu2 ** 2
    mu1_mu2 = mu1 * mu2

    sigma1_sq = gaussian_filter(img1 ** 2, sigma) - mu1_sq
    sigma2_sq = gaussian_filter(img2 ** 2, sigma) - mu2_sq
    sigma12 = gaussian_filter(img1 * img2, sigma) - mu1_mu2

    # 计算SSIM
    numerator = (2 * mu1_mu2 + C1) * (2 * sigma12 + C2)
    denominator = (mu1_sq + mu2_sq + C1) * (sigma1_sq + sigma2_sq + C2)

    ssim_map = numerator / denominator

    # 返回平均SSIM
    return np.mean(ssim_map)

# 简化版实现（使用scikit-image）
from skimage.metrics import structural_similarity
```

```
54
55  def calculate_ssim_simple(img1, img2):
56      """使用scikit-image计算SSIM"""
57      return structural_similarity(img1, img2,
58                                   data_range=255,
59                                   multichannel=True)
```

课堂动手（无需写入实验报告）

1. **比较不同显著性图的效果**

   使用我们提供的三种显著性图算法调整缩放比例和图片，比较每种算法对不同类型图片（风景、人像等）的效果

2. **实现CNN显著性图（可选）**

   使用预训练的卷积神经网络来提取图像特征并可视化

平时作业（需撰写实验报告）

使用神经网络来增强重定向质量

题目描述

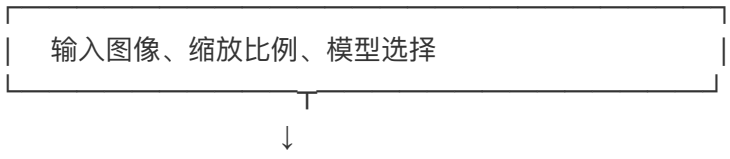
随着多种显示设备的普及（手机、平板、电脑、电视），图像需要适应不同的宽高比。传统的缩放方法会导致内容变形或重要信息丢失。Seam Carving是一种内容感知的图像缩放技术，但原始算法缺乏对图像语义内容的理解。本实验将探索如何使用轻量级深度学习模型增强Seam Carving算法，在有限的计算资源下实现更智能的图像重定向。本题目要求你结合卷积神经网络来增强显著性图效果，使用Seam Carving算法进行图片重定向，并观察效果和定量计算前后PSNR、SSIM得分。

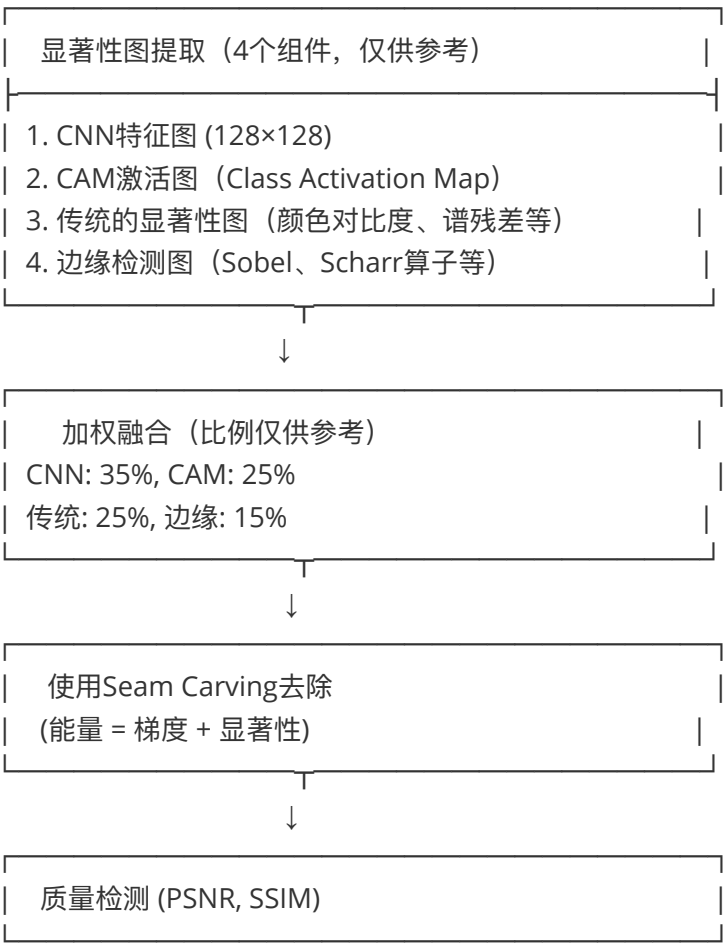
要求

1. 掌握常规显著性检测和CNN显著性图在图像重定向中的作用，考虑结合多种显著性图来增强特征;
2. 理解并实现基础Seam Carving算法，利用上一步的显著性图进行重定向;
3. 计算使用插值算法和基于显著性图的重定向算法应用后的SSIM和PSNR得分，列出表格进行比较。

请在2025年10月20日前完成，请打包**代码**、**实机演示视频**和**实验报告**到一个zip文件中并发送邮件到cvexp2025@163.com。邮件需命名为：学号-姓名-第四次作业，且**附件不超过30M**

参考思路




显著性图提取（4个组件，仅供参考）
1. CNN特征图 (128×128)
2. CAM激活图（Class Activation Map）
3. 传统的显著性图（颜色对比度、谱残差等）
4. 边缘检测图（Sobel、Scharr算子等）
↓
加权融合（比例仅供参考）
CNN: 35%, CAM: 25%
传统: 25%, 边缘: 15%
↓
使用Seam Carving去除
(能量 = 梯度 + 显著性)
↓
质量检测 (PSNR, SSIM)