基于 Python 的智能缩放、图像分割与区域分类一体化

汇报人：张书旋

Python 课程作业验收

2025.06

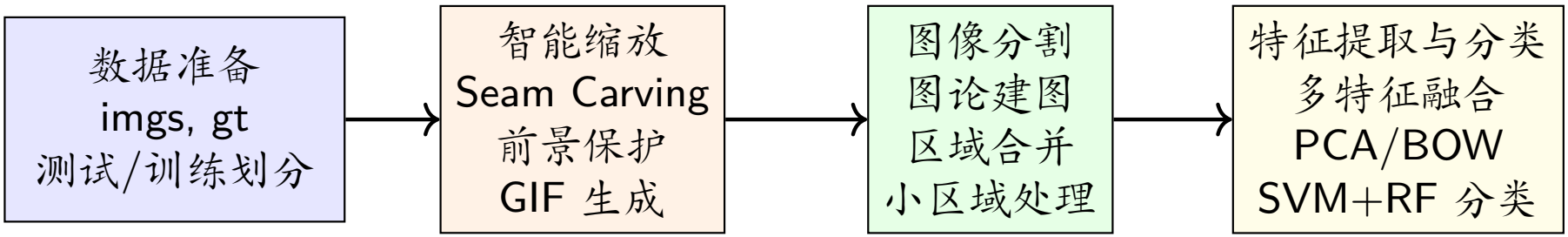
选题理由

- 随着人工智能技术的快速发展，图像的智能处理在医学影像、自动驾驶等领域有着广泛的应用需求。传统的图像缩放方法容易导致重要内容变形或丢失；而单一特征的图像分割与区域分类方法在复杂场景下准确率有限。
  通过一体化的系统设计，既能有效保护图像中的前景和重要区域，又能对分割区域进行高效准确的分类，具有较强的创新性和应用价值。

- 涵盖了数据处理、图像分割、特征提取、特征选择、机器学习分类、结果评估等多个环节，而 Python 拥有丰富的图像处理和机器学习库，该项目能够有效提升使用 Python 各种库和算法实现能力。

- 项目不仅实现了传统的图像分割算法，还融合了颜色、纹理、形状等多种特征，并采用 SVM、随机森林等集成学习方法进行区域分类，体现了传统方法与机器学习的结合与创新。

项目流程


数据准备
imgs, gt
测试/训练划分
智能缩放
Seam Carving
前景保护
GIF 生成
图像分割
图论建图
区域合并
小区域处理
特征提取与分类
多特征融合
PCA/BOW
SVM+RF 分类

数据集介绍

数据集：

- 来源：
  HKU-IS
- 介绍：
  1000 组图片，每组包括一张原图和其前景掩码

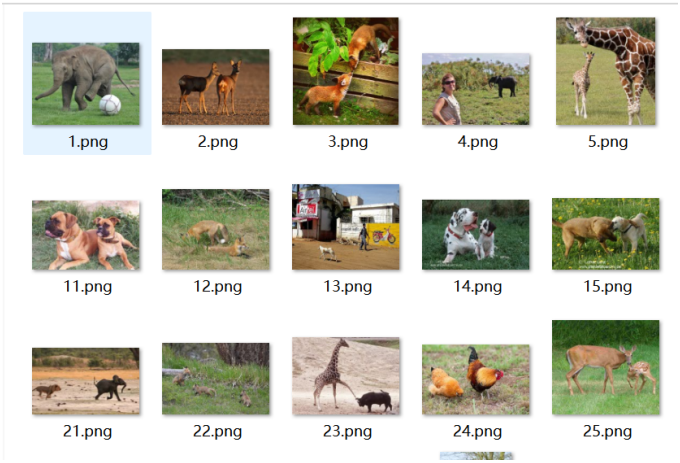


图 1: 原始图片

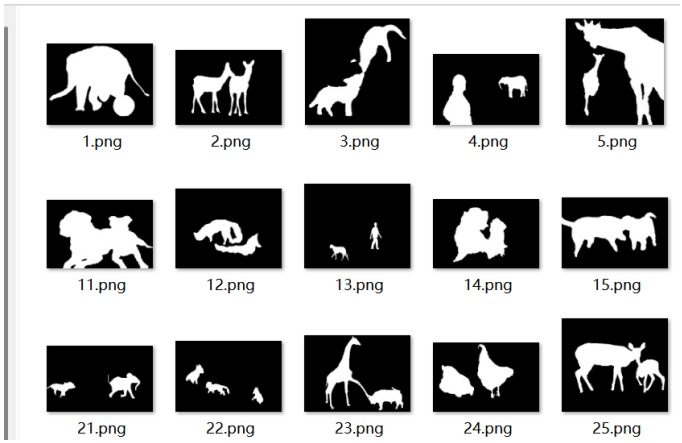


图 2: 对应前景掩码

3.1.1 图像缩放模块

功能实现：

- 输入：原始图像 + 掩码 (mask)
- 输出：调整大小后的图像
- 处理流程：
  1. 能量图计算
  2. 动态规划
  3. 路径选择
  4. 图像重构

核心算法：

- 能量图计算：基于 RGB 差异
- 动态规划：寻找最小能量路径
- 路径选择：水平和垂直方向
- 图像重构：移除选定的 seam

3.1.2 动态规划实现

```python
    def getDynamicEnergyMap(self, img, mask):

        height, width = img.shape
        img = img * mask  # 用Mask处理图像
        paddingImg = np.pad(img, ((1, 1), (1, 1)), 'constant', constant_values=0)  # 图像外围填充一圈0
        dynamicEnergyMap = np.zeros((height + 2, width + 2), dtype=np.float32)  # 使用float32
        routeMap = np.ones((height, width), dtype=np.int32) * -1
        for hIndex in range(height):  # 从上到下逐行处理
            M = np.zeros((width, 3), dtype=np.float32)  # 使用float32
            # 采用 Forward Seam Removing 机制进行计算
            M[:, 0] = dynamicEnergyMap[hIndex, :-2] + np.abs(paddingImg[hIndex][1:-1] - paddingImg[hIndex + 1][:-2])
            M[:, 1] = dynamicEnergyMap[hIndex, 1:-1]
            M[:, 2] = dynamicEnergyMap[hIndex, 2:] + np.abs(paddingImg[hIndex][1:-1] - paddingImg[hIndex + 1][2:])
            colEnergy = np.abs(paddingImg[hIndex + 1][2:] - paddingImg[hIndex + 1][:-2])
            dynamicEnergyMap[hIndex + 1, 1:-1] = np.min(M, axis=1)
            routeMap[hIndex] = np.argmin(M, axis=1) - 1
            # 处理边界点
            # 像素(i,0)只能选择(i-1,0)和(i-1,1)
            if M[0, 1] <= M[0, 2]:
                routeMap[hIndex, 0] = 0
                dynamicEnergyMap[hIndex + 1, 1] = M[0, 1]
            else:
                routeMap[hIndex, 0] = 1
                dynamicEnergyMap[hIndex + 1, 1] = M[0, 2]
            # 像素(i,-1)只能选择(i-1,-1)和(i-1,-2)
            if M[-1, 0] <= M[-1, 1]:
                routeMap[hIndex, -1] = -1
                dynamicEnergyMap[hIndex + 1, -1] = M[-1, 0]
            else:
                routeMap[hIndex, -1] = 0
                dynamicEnergyMap[hIndex + 1, -1] = M[-1, 1]
            dynamicEnergyMap[hIndex + 1, 1:-1] += colEnergy  # 计算最终的energy
        return dynamicEnergyMap[1:-1, 1:-1], routeMap
```

图 3: 核心代码

3.1.3 智能图像缩放

原始图像

图 4: 输入图像

最终结果

图 5: 缩放后的图像

Seam 路径

图 6: 移除的 seam 路径

输出说明：

- 19.png：原始图像像素：200*150
- 19_final.png：缩放后的最终结果像素：137*103
- 19_final2.png：标记了所有被移除的 seam 路径

3.2.1 图像分割模块

功能实现：

- 输入：缩放后图像
- 输出：分割后的区域标记图
- 处理流程：
  1. 图像预处理：RGB 转灰度
  2. 图构建：计算像素间距离
  3. 区域分割：基于阈值合并
  4. 小区域处理：移除 <50 像素区域

3.2.2 基于阈值合并

```python
def segment_graph(sorted_graph, num_node, k):
    '''
    对图结构进行分割
    :param sorted_graph: 根据权重对所有的边排序后的图结构
    :param num_node: 节点总数
    :param k: 控制聚类程度的阈值
    :return: 分割后的结果
    '''
    res = area(num_node)
    # 记录各个区域的类内不相似度
    threshold = [k] * num_node
    for edge in sorted_graph:
        u = res.find_parent(edge[0])
        v = res.find_parent(edge[1])
        w = edge[2]
        # 如果边接连的两点不属于同一区域
        if u != v:
            # 如果边的权重小于阈值
            if w <= threshold[u] and w <= threshold[v]:
                # 合并两个节点
                res.merge2node(u, v)
                parent = res.find_parent(u)
                # 更新最大类内间距
                threshold[parent] = np.max([w, threshold[u], threshold[v]]) + k / res.size[parent]
    return res
```

图 7: 核心代码

3.2.3 图像分割模块主要函数与作用

主要函数及其作用

- **area**：并查集结构，管理像素归属区域，实现区域合并与查找。
- **create_edge**：计算像素间的 RGB 距离，生成图的边及权重。
- **build_graph**：为整张图像建立像素连边的图结构。
- **segment_graph**：基于动态阈值的区域合并，实现图像分割的核心算法。
- **remove_small_area**：将小于阈值的区域合并到相邻大区域，避免碎片化。
- **generate_image**：为每个分割区域随机上色，生成分割可视化图。
- **cal_IOU**：计算分割结果与真实掩码的 IOU 指标，生成前/背景二值图。

```
# 图像处理
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
height, width, channel = img.shape
img = np.asarray(img, dtype=float)
# 分开组成grb
img = cv2.split(img)
# 建立图结构
graph = build_graph(img, width, height)
sorted_graph = sorted(graph, key=weight)  # 根据权重对所有的边进行排序
# 分割
res = segment_graph(sorted_graph, width * height, k[index])
res = remove_small_area(res, sorted_graph, min_size)
# 生成结果图
areaset = []
img = generate_image(res, width, height, areaset)
IOU, img2 = cal_IOU(img_gt, res, width, height, areaset)
print(f"  IOU: {IOU}")

# 保存结果图片
cv2.imwrite(os.path.join(RESULT_DIR, f"{img_number}_result.png"), img)
cv2.imwrite(os.path.join(RESULT_DIR, f"{img_number}_mask.png"), img2)
```

图 8: 可视化和分割效果评估

3.2.4 图像分割运行结果

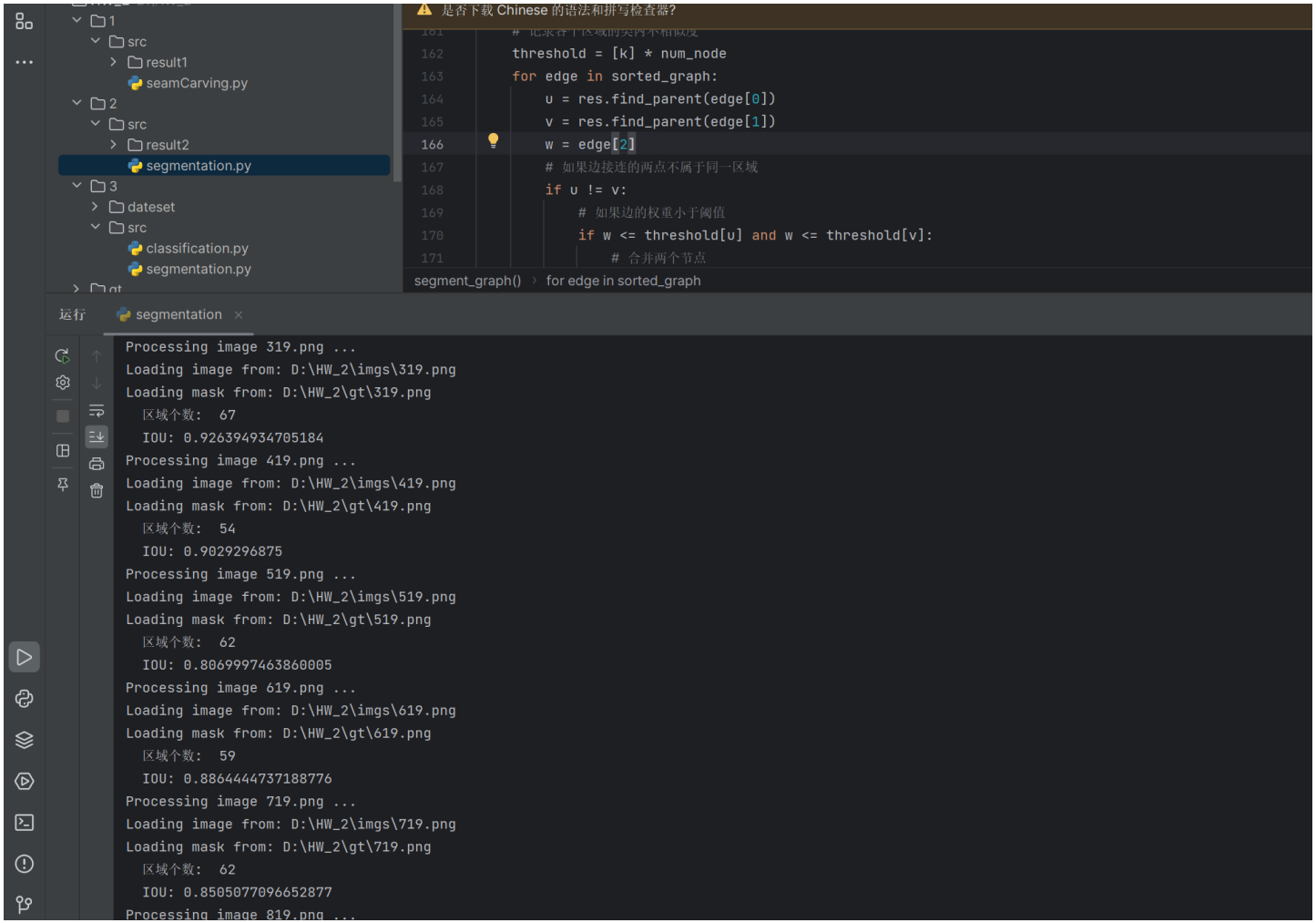


图 9: 运行结果截图

3.2.5 智能图像分割

处理图像(.png)	原图	Mask	标记结果	处理结果	IOU	区域总数
19					0.7961	69
119					0.8188	66
219					0.6891	59
319					0.9264	67
419					0.9029	54

3.3.1 区域分类模块

功能实现：

- 输入：分割后的区域
- 输出：前景/背景分类结果
- 处理流程：
  - 特征提取
  - 特征选择
  - 模型训练
  - 区域分类

核心算法：

- 多特征融合：颜色、纹理、形状
- 集成分类：SVM + 随机森林
- PCA 降维、BOW 特征

3.3.2 特征工程

```python
2 用法
def cal_fmat(img, bin, res, t):
    '''
    计算特征矩阵，获取区域颜色直方图和全图颜色直方图，并进行拼接
    :param img: 待处理图像
    :param bin: 每种颜色的维度
    :param res: 图像分割后的结果
    :param t: 一维坐标到二维坐标的映射关系
    :return: 各个区域的特征矩阵
    '''
    height, width, _ = img.shape
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # 提取归一化全图RGB颜色直方图
    district_histogram = generate_histogram(img, bin)
    m = []
    # 图像分割后的每一个区域
    for comp in res.all_area():  # 遍历图像分割后每一个区域的根节点
        mask = np.zeros(shape=(height, width), dtype=np.uint8)
        v = res.one_area(comp)  # 找到该区域的所有像素点
        # 制作覆盖该区域的mask
        for i in v:
            pix = t.t[i]
            mask[pix[0], pix[1]] = 255
        full_histogram = generate_histogram(img, bin, mask)
        # 新增：提取区域的LBP特征
        region = cv2.bitwise_and(img_gray, img_gray, mask=mask)
        region_uint8 = region.astype(np.uint8)
        lbp_hist = extract_lbp(region_uint8)
        glcm_feat = extract_glcm_features(region_uint8)
        hog_feat = extract_hog(region_uint8)
        shape_feat = extract_shape_features(mask)
        # 拼接颜色直方图、全图直方图和LBP特征
        fvec = np.concatenate((full_histogram, district_histogram, lbp_hist, glcm_feat, hog_feat, shape_feat))
        m.append(fvec)
    m = np.array(m)
    return m
```

图 11: 特征提取与融合

```python
# 处理训练集
# PCA降维
x_train = PCA(n_components=20).fit_transform(x_train)
# BOW特征
trainer1 = cv2.BOWKMeansTrainer(50)  # 使用K-Means聚类方法获得50个聚类中心点
trainer1.add(np.float32(x_train))
```

图 12: 特征降维与选择

3.3.3 实验结果

```python
def test(x_train, y_train, x_test, y_test):
    # 自动调参SVM
    svm_best = auto_tune(x_train, y_train)
    # 随机森林
    rf = RandomForestClassifier(n_estimators=100, random_state=42, max_depth=5)
    # 集成分类器
    clf = VotingClassifier(estimators=[('svm', svm_best), ('rf', rf)], voting='soft')
    clf.fit(x_train, y_train)
```

图 13: 分类模型训练与预测

```
运行 classification
Processing 990.png ...
Processing 995.png ...
Processing 1000.png ...
Test Set:
Processing 19.png ...
Processing 119.png ...
Processing 219.png ...
Processing 319.png ...
Processing 419.png ...
Processing 519.png ...
Processing 619.png ...
Processing 719.png ...
Processing 819.png ...
Processing 919.png ...
Selected feature indices: [ 0  2  4  5  6  9 12 13 15 16 17 20 21 22 24 25 27 28 29 30 31 32 33 35
 37 38 40 41 42 43 44 45 47 48 49 50 51 53 54 55 56 57 58 59 64 65 66 67
 68 69]
开始RFE特征选择...
RFE特征选择完成。RFE选中特征索引： [17 19 21 26 27 28 37 38 42 46]
开始SVM自动调参...
SVM自动调参完成。SVM最优参数： {'svc__C': 1, 'svc__gamma': 0.1, 'svc__kernel': 'rbf'}
5折交叉验证准确率： [0.80074161 0.80089791 0.80070271 0.80070271 0.80070271]
平均准确率： 0.8007495318521134
训练集的准确率： 0.8007885696439725
测试集的准确率： 0.7862537764350453
```

项目总结

项目亮点：

- 如何在 Seam Carving 过程中有效避免前景区域被误删
- 如何用图论方法动态合并像素区域，兼顾分割精度与速度
- 如何从分割区域提取多种特征并有效降维，提升分类效果
- 如何让分割、特征提取、分类等多环节高效衔接，自动处理大批量数据

主要收获：

- 掌握了图像分割与区域分类主流算法
- 熟悉特征工程与集成学习实践
- 提升了 Python 图像处理与机器学习能力

改进与展望

存在的问题：

- 部分边界分割不够精细
- 小区域分类易受噪声影响
- 特征选择和参数调优有待加强

未来展望：

- 尝试深度学习方法提升精度
- 引入更多高阶特征
- 优化算法效率，扩展应用场景

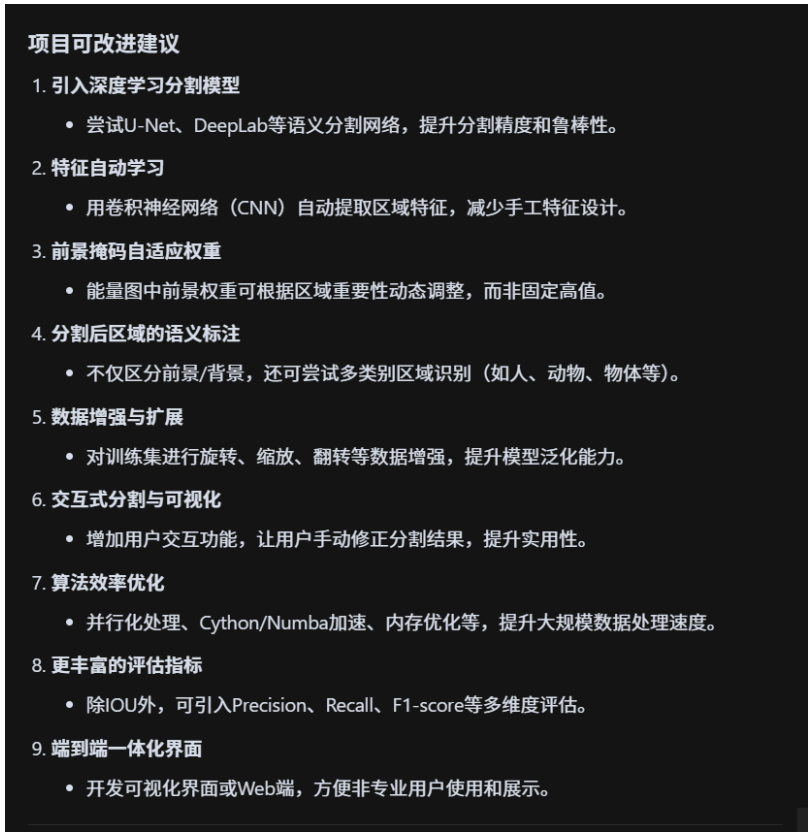


图 15: 改进

Thanks!