Project 5 Writeup

Implementation Detail

This project implements selective search algorithm for object recognition. This algorithm applies hierarchical segmentation and is mainly divided into 3 parts. Step 1 is generate initial sub-segmentation. Step 2 is extract all the regions based on four similarities: color, texture, size and fill, then recursively combine similar regions into larger ones. Step 3 is use the generated regions to produce candidate object locations.

Result

1. Figure 1 shows that each test image can be efficiently detected.
2. In this experiment, we set the same weights of all 4 similarities, but actually it shall be modified corresponds to different cases.

   For example, in result 6 (Figure 1, middle right), obviously, we can increase the weight of texture and decrease the weight of color. As the figure 2 shows, the desired effect is better.

Extra Credit (Optional)

1. Generate initial sub-segmentation using felzenszwalb algorithm and merge the image mask to the image as a 4th channel

```
def generate_segments(im_orig, scale, sigma, min_size):
    segement_mask = segmentation.felzenszwalb(im_orig, scale,
                                               sigma, min_size)
    seg_img = np.zeros((im_orig.shape[0], im_orig.shape[1], 4))
    seg_img[:, :, :3] = im_orig
    seg_img[:, :, 3] = segement_mask
    return seg_img
```

2. Calculation 4 similarities based on color, texture, size and fill

```
def sim_colour(r1, r2):
    """
    2.1. calculate the sum of histogram intersection of colour
    """
    sum = 0
    for a, b in zip(r1["hist_color"], r2["hist_color"]):
        con = np.concatenate(([a], [b]), axis=0)
        sum += np.sum(np.min(con, axis=0))
```

Figure 1: 9 different test images (3 from each of Art History, Class. Arch and Chris. Arch)

```python
    return sum


def sim_texture(r1, r2):
    """
    2.2. calculate the sum of histogram intersection of texture
    """
    sum = 0
    for a, b in zip(r1["hist_text"], r2["hist_text"]):
        con = np.concatenate(([a], [b]), axis=0)
        sum += np.sum(np.min(con, axis=0))
    return sum


def sim_size(r1, r2, imsize):
    """
    2.3. calculate the size similarity over the image
    """
    return 1 - (r1["size"] + r2["size"]) / imsize
```

Figure 2: *Left:* co-weights *Right:* higher texture weight

```
def sim_fill(r1, r2, imsize):
    """
    2.4. calculate the fill similarity over the image
    """
    max_x = max(r1["max_x"], r2["max_x"])
    min_x = min(r1["min_x"], r2["min_x"])
    max_y = max(r1["max_y"], r2["max_y"])
    min_y = min(r1["min_y"], r2["min_y"])
    bb_size = (max_x - min_x) * (max_y - min_y)
    return 1 - (bb_size - r1["size"] - r2["size"]) / imsize


def calc_sim(r1, r2, imsize):
    return (sim_colour(r1, r2) + sim_texture(r1, r2)
            + sim_size(r1, r2, imsize) + sim_fill(r1, r2, imsize)
                                                )
```

3. extract regions and neighbours

```
def extract_regions(img):
    R = {}
    hsv_img = rgb2hsv(img[:, :, :3])
    mask = img[:, :, 3]
    texture_gradient = calc_texture_gradient(img[:, :, :-1])
    for i in range(int(np.max(mask))):
        y_set, x_set = np.where(mask == i)
        masked_area = mask == i
        R[i] = {
            "labels": i,
            "size": np.sum(masked_area == 1),
            "min_x": np.min(x_set),
            "max_x": np.max(x_set),
            "min_y": np.min(y_set),
            "max_y": np.max(y_set),
            "hist_color": calc_colour_hist(hsv_img[masked_area]),
            "hist_text": calc_texture_hist(texture_gradient[
                                                masked_area])
        }
    return R

def extract_neighbours(regions):
```

```
    def intersect(a, b):
        if (a["min_x"] < b["min_x"] < a["max_x"]
            and a["min_y"] < b["min_y"] < a["max_y"]) or (
                a["min_x"] < b["max_x"] < a["max_x"]
                and a["min_y"] < b["max_y"] < a["max_y"]) or (
                a["min_x"] < b["min_x"] < a["max_x"]
                and a["min_y"] < b["max_y"] < a["max_y"]) or (
                a["min_x"] < b["max_x"] < a["max_x"]
                and a["min_y"] < b["min_y"] < a["max_y"]):
            return True
        return False

    # Hint 1: List of neighbouring regions
    # Hint 2: The function intersect has been written for you and
                                           is required to check
                                          neighbours
    neighbours = []
    for i in range(len(regions)):
        for j in range(len(regions)):
            if i in regions.keys() and j in regions.keys() and i
                                                    < j:
                if intersect(regions[i], regions[j]):
                    n1 = (i, regions[i])
                    n2 = (j, regions[j])
                    neighbours.append((n1, n2))
    return neighbours
```

4. merge regions

```
def merge_regions(r1, r2):
    new_size = r1["size"] + r2["size"]
    rt = {
        "labels": r1["labels"] + r2["labels"],
        "size": new_size,
        "min_x": min(r1["min_x"], r2["min_x"]),
        "min_y": min(r1["min_y"], r2["min_y"]),
        "max_x": max(r1["max_x"], r2["max_x"]),
        "max_y": max(r1["max_y"], r2["max_y"]),
        "hist_color": (r1["hist_color"] * r1["size"] + r2["
                                    hist_color"] * r2["
                                    size"]) / new_size,
        "hist_text": (r1["hist_text"] * r1["size"] + r2["
                                    hist_text"] * r2["size
                                    "]) / new_size
    }
    return rt
```

5. Hierarchical search for merging similar regions

```
    while S != {}:
        # Get largest similarity
        i, j = sorted(S.items(), key=lambda i: i[1])[-1][0]

        # Task 4: Merge corresponding regions.
        t = max(R.keys()) + 1.0
        R[t] = merge_regions(R[i], R[j])
```

```
        # Task 5: Mark similarities for regions to be removed
        reg2remove = []
        for key in S.keys():
            if (i in key) or (j in key):
                reg2remove.append(key)

        # Task 6: Remove old similarities of related regions
        for k in reg2remove:
            del S[k]

        # Task 7: Calculate similarities with the new region
        for (x, y) in reg2remove:
            if (x, y) != (i, j):
                if x != i:
                    new = x
                elif y != j:
                    new = y
                S[(t, new)] = calc_sim(R[t], R[new], imsize)
```

6. Generating the final regions from R

```
for r in R.values():
    regions.append({ 'rect': (x, y, w, h),
                     'size': r['size'], 'labels': r['labels'] })
```