Project 5 Writeup

Result

Normal implement

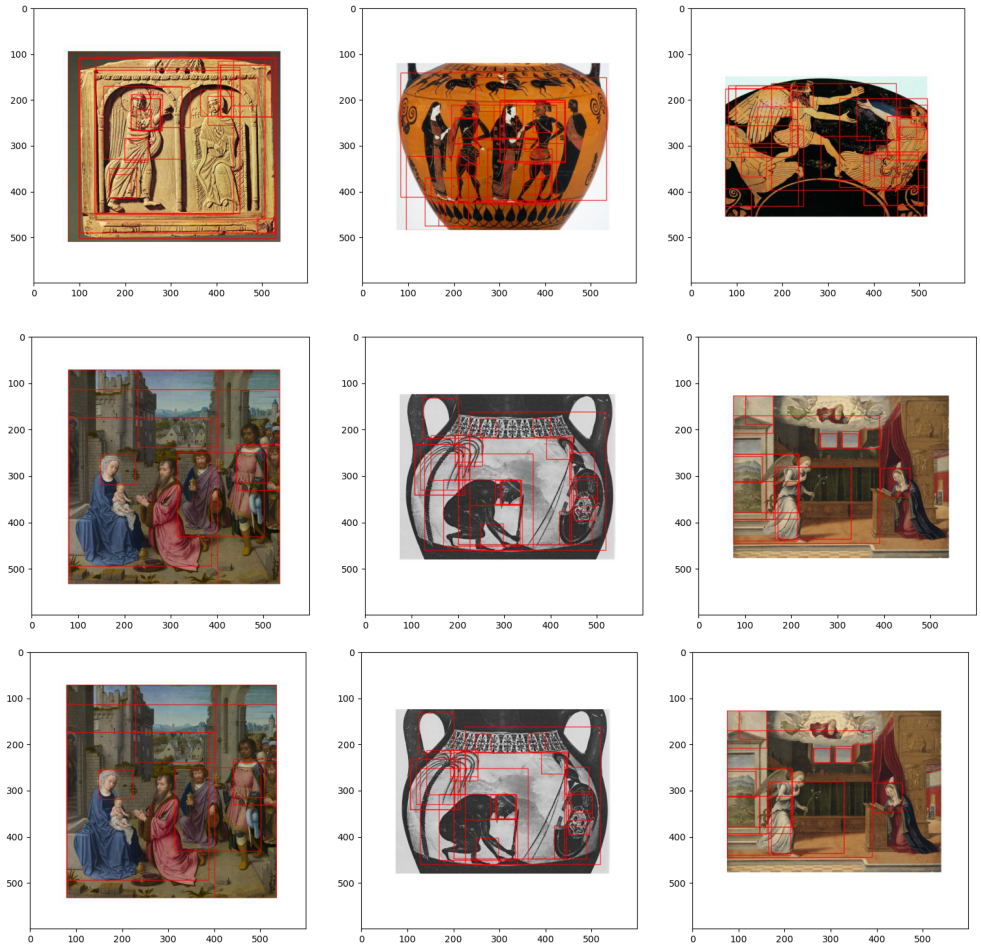

Figure 1: Result with 9 images

As shown in Figure 1, after implementation, all results have reached the expected results.

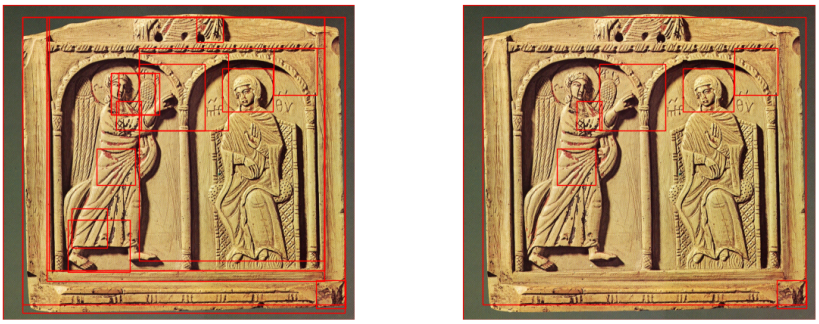

Figure 2: *Left:* Origin result. *Right:* Norm modified result.

Play around with it

Remove normalize

As shown in Figure 2 , we remove the normalize part of the method *calc_texture_hist* and *calc_color_hist*

The code shown on below:

```
def calc_colour_hist(img):
  BINS = 25
  hist = []
  for c in range(3):
  h,_ = np.histogram(img[:,c],bins=BINS)
  hist.append(h)
  return np.array(hist) / len(img)

def calc_texture_hist(img):
  BINS = 10
  hist = []
  for c in range(3):
  h,_ = np.histogram(img[:,c],bins=BINS)
  hist.append(h)
  return np.array(hist) / len(img)
```

That makes the algorithm generate less region than the origin one.

We have no convincing answer to this. Our guess is that, the histogram is out of range. Caused the problem

Parameter of Selective search

As shown in Figure 3 , we change the *scale* parameter from 500 to 1000.

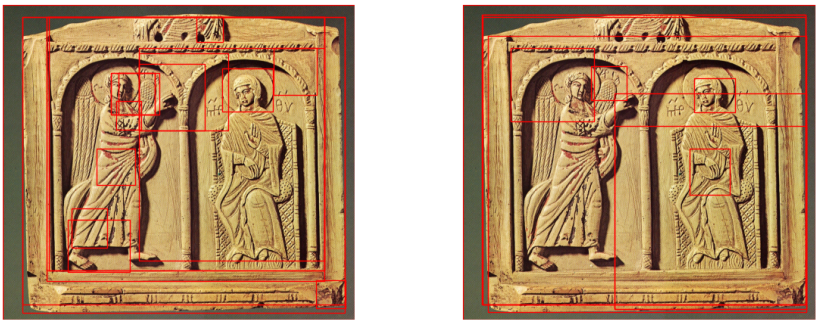

Figure 3: *Left:* Origin result. *Right:* Scale modified result.

This is very reasonable, after expanding scale. The region of the mask returned by the *felzenszwalb* method becomes larger. The number of regions after fusion becomes smaller.

The same explanation applies to the parameter *min_size*, result shown on Figure 4**??**. The parameter has bin expand from 20 to 40.

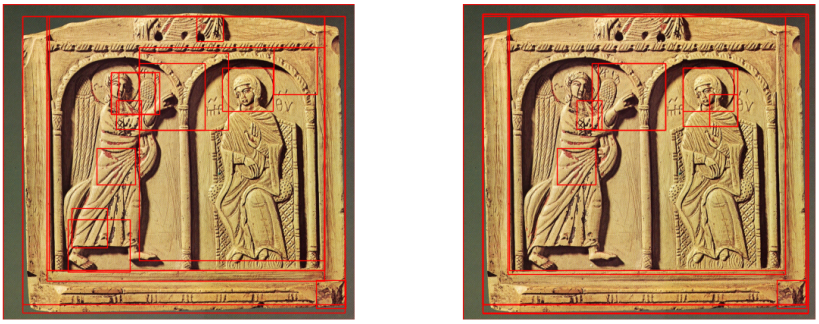

Figure 4: *Left:* Origin result. *Right:* Min_size modified result.

Parameter of bins

As shown in Figure 5, we increased the parameter *bins* of texture histogram from 10 to 20. As a result, the algorithm is more sensitive to the characteristics of the problem, so fewer regions are generated.

Since the color interval of *annun3* is small, when changing the parameter *bins* of color

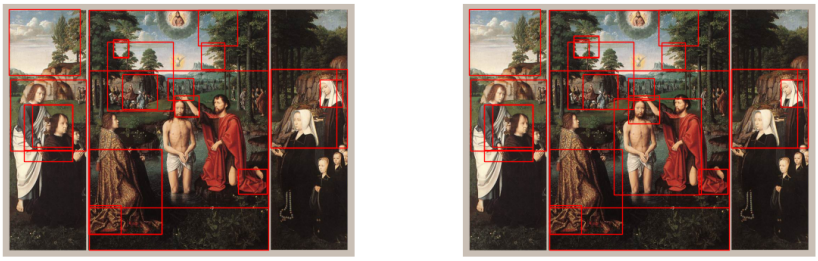

Figure 5: *Left:* Origin result. *Right:* Texture bins modified result.

histogram. We choose *baptism1*, which is richer in color, as a reference. The result shown on Figure 6

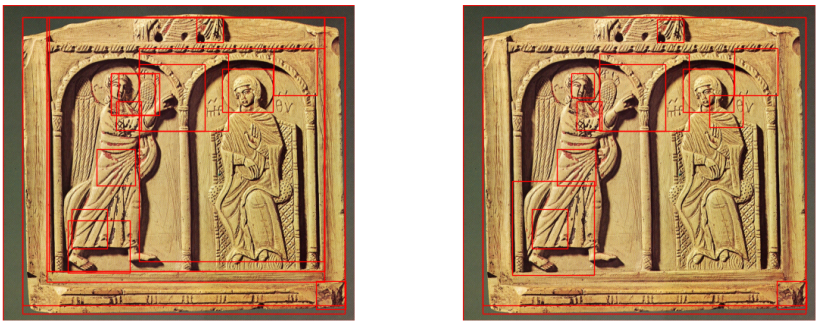

Figure 6: *Left:* Origin result. *Right:* Color bins modified result.

Extra Credit (Optional)

Show the implementation of selective search

1. Use the existing feltszwalb of the skimage library to generate a mask and add it to the original image as the fourth channel

```
def generate_segments(im_orig, scale, sigma, min_size):
    mask = felzenszwalb(im_orig, scale, sigma, min_size)
    mask = np.expand_dims(mask,2)
    im_orig = np.concatenate((im_orig,mask),axis=2)
    return im_orig
```

2. Generate regions according to the prompts

```
def extract_regions(img):
    # to hsv and get the mask
    im_hsv = rgb2hsv(img[:,:,:-1])
    mask = img[:,:,-1]
    # calc texture for the image
    im_text = calc_texture_gradient(img[:,:,:-1])
    for i in range(np.max(mask)):
        # get all idx of the mask
        y_all,x_all = np.where(mask==i)
        # get the current mask
        currentMask = mask==i
        # put every thing in to the dict
        R[i] = {
          "labels": i,
          "min_x": np.min(x_all),
          "max_x": np.max(x_all),
          "min_y": np.min(y_all),
          "max_y": np.max(y_all),
          "colorHist": calc_colour_hist(im_hsv[currentMask]),
          "textureHist": calc_texture_hist(im_text[currentMask]),
          "size": len(im_hsv[currentMask])
        }
      return R


def calc_colour_hist(img):
BINS = 25
hist = []
for c in range(3):
    h,_ = np.histogram(img[:,c],bins=BINS)
    hist.append(h)
    return np.array(hist) / len(img)

def calc_texture_gradient(img):
ret = np.zeros((img.shape[0], img.shape[1], img.shape[2]))
for c in range(3):
    ret[:,:,c] = local_binary_pattern(img[:,:,c],P=8,R=1)
    return ret

def calc_texture_hist(img):
    BINS = 10
    hist = []
    for c in range(3):
        h,_ = np.histogram(img[:,c],bins=BINS)
        hist.append(h)
    return np.array(hist) / len(img)
```

3. Whether the detection area overlaps

```
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

    neighbours = []

    for r_a in regions.items():
        for r_b in regions.items():
            if intersect(r_a[1],r_b[1])and r_a[0]<r_b[0]:
                neighbours.append((r_a,r_b))
    return neighbours
```

4. compute similarity

```
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))


def sim_colour(r1, r2):
    # get the smaller value of both color hist
    cHist_1 = r1["colorHist"]
    cHist_2 = r2["colorHist"]
    res = np.sum(cHist_1[cHist_1<=cHist_2]) + np.sum(cHist_2[
                                    cHist_1>cHist_2])
    return res


def sim_texture(r1, r2):
    # get the smaller value of both texture hist
    tHist_1 = r1["textureHist"]
    tHist_2 = r2["textureHist"]
    res = np.sum(tHist_1[tHist_1<=tHist_2]) + np.sum(tHist_2[
                                    tHist_1>tHist_2])
    return res


def sim_size(r1, r2, imsize):
    # implement the form
    return 1 - (r1["size"] + r2["size"]) / imsize


def sim_fill(r1, r2, imsize):
    # implement the form
    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"])
    bbSize = (max_x - min_x) * (max_y - min_y)
    return 1 - (bbSize - r1["size"] - r2["size"]) / imsize
```

5. merge the regions

```
def merge_regions(r1, r2):
    new_size = r1["size"] + r2["size"]
    rt = {}

    rt["size"] = new_size
    rt["max_x"] = max(r1["max_x"], r2["max_x"])
    rt["min_x"] = min(r1["min_x"], r2["min_x"])
    rt["max_y"] = max(r1["max_y"], r2["max_y"])
    rt["min_y"] = min(r1["min_y"], r2["min_y"])
    # weighted average of chist
    rt["colorHist"] = (r1["colorHist"] * r1["size"] + r2["
                                    colorHist"] * r2["size"]) /
                                    new_size

    # weighted average of thist
    rt["textureHist"] = (r1["textureHist"] * r1["size"] + r2["
                                    textureHist"] * r2["size"]) /
                                    new_size
    rt["labels"] = r1["labels"] + r2["labels"]
    return rt
```

6. After fusion, recalculate the similarity

```
# Task 5: Mark similarities for regions to be removed

temp = []
for k in S.keys():
    if i in k or j in k:
        temp.append(k)

# Task 6: Remove old similarities of related regions

for k in temp:
    del S[k]

# Task 7: Calculate similarities with the new region

for (a,b) in temp:
    if (a,b) != (i,j):
        if a != i:
            current = a
        elif b != j:
            current = b
        S[(t, current)] = calc_sim(R[t], R[current], imsize)
```

7. final regions

```
regions = []
for r in R.values():
    region = {}
```

```
    region['rect'] = (r["min_x"], r["min_y"], r["max_x"] - r["
                                    min_x"], r["max_y"] - r["min_y
                                    "])
    region['labels'] = r["labels"]
    region['size'] = r["size"]
    regions.append(region)
```