Lec4_Feature_Extraction

Features and Quantization

Hair Color, Skin Color in number → feature vector - quantization

→ Feature Space & grouping together

Metrics (Distance and Similarity)

inner product: $<x, y>=\Sigma_i x_i y_i$

cosine similarity: $cos(\theta)=<x, y>/(|x||y|)$

Euclidian Distance: $d(x, y)=|x-y|=\sqrt(<x-y, x-y>)$

Global and Local Features (Color Histograms, LBP, SIFT)

Global img feature

- Color Histogram(color feature): distribution of composition of colors, number of pix in (RGB)
  - **color quantization**: color space divided to small color intervals, and each cell becomes a bin of the histogram. ⇒ x-RGB, y-freq
- HOG - Histogram of Orientated Gradient(shape feature)
- LBP - Local Binary Pattern(texture feature): internal charac of object surface, structure & arrangement of object surface, its relationship with surrounding objects.
  - apply: Texture/face description, Medical img analysis, Pedestrian detection, Background modeling, few-shot learning

$$LBP(P, R)=\Sigma_i S(g_p-g_c)*2(p-1)$$
$$S(x)=1 \text{ if } x\geq 0 \ / \ S(x)=0 \text{ if } x<0$$

  - feature vector X=[x1,x2,x3,.., xk] disadvantage: coverage area is fixed
- Circle LBP:
  1. Given the center point (x_c,y_c), sampling point location can be calculated by: ... in code

$$x_p=x_c+Rcos\frac{2\pi\times p}{P}, y_p=y_c-Rsin\frac{2\pi\times p}{P}$$

Local img feature

- SIFT(Scale-invariant feature transform SIFT)

1. Candidate Localization: Find Scale-invariant candidates/key candidate

- find scale - best at its center (key point),
- scale-invariant: no matter what original img scale is, same best scale will found
- Rescale and blur, Difference of Gaussian(local max & min stand out neighbor

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

$$D(x, y, \sigma) = L(x, y, \sigma) - L(x, y, k\sigma) = (G(x, y, \sigma) - G(x, y, k\sigma) * I(x, y)$$

2. Refinement: Keypoint filtering: remove low-contrast position / on edge
   - edge candidate: principal curvature across edge is larger than along edge
   - Hessian Matrix $H=[Dxx Dxy][Dxy Dyy]$ remove the edge cand
3. Orientation Assignment: Estimate orientations for keypoints
   - gradient magnitudes and orientations in small window around keypoint – at appropriate scale.
   - Assign the dominant orientation as the orientation of the keypoint.
   - separate descriptor for multiple peaks
4. Description Generation: Build description
   - a small region around keypoint, divide n*n cells, build **gradient orientation histogram,** weighted by gradient magnitude
   - img gradients ⇒ keypoint descriptor(rn^2), length of descripter is 128, r bins (direction)=8 n(sub-win) =4
   - size of window adjust as scale of keypoint

```
**Gray Histogram**
def histogram(img, norm: bool = True):
        ****hist = np.array([len(img[img == i]) for i in range(256)])
        if norm:
            return hist / np.size(img)
        return hist
****def patch_std(img : np.array, n_divide : int = 4) -> float:
        patches = [j for i in np.array_split(img, n_divide, axis=0)
                 for j in np.array_split(i, n_divide, axis=1)]
    return np.mean(list(map(np.std, patches))) // **Calculate std**

**RGB-Color Histogram**
np.array([gray_histogram(img[..., i], norm=norm) for i in range(3)])

**LBP pattern**
def unfold(img : np.array, ksize : int = 3) -> np.array:
    # unfold without center point, only odd kernel size is supported
        img: An image with size of H x W.
        ksize: The kernel size
    assert ksize % 2 == 1
    assert img.ndim == 2
```

```
    H, W = img.shape

    # Expand the orignal image's shape for better moving. For the third channel, 
we pre-define its dimension as eight,
    # the LBP result of each pixel depends on the values of the surrounding 8 
points
    target = np.zeros((H+ksize-1, W+ksize-1, ksize**2-1), dtype=img.dtype)
    n = 0
    for h in range(ksize):
        for w in range(ksize):
            if h == ksize // 2 and w == ksize // 2:
                continue
            target[h:h+H, w:w+W, n] = img
            n += 1
    return target[ksize//2:ksize//2+H, ksize//2:ksize//2+W, :]
def original_LBP(img : np.array) -> np.array:
    # calculate the original version of LBP
    - img:An image with size of H x W.
    - Pattern:[[4 3 2],[5 None 1][6 7 8]]
    img_unfold = unfold(img)
    factor1 = img_unfold >= img[..., None]
    factor2 = np.array([128, 64, 32, 1, 16, 2, 4, 8], dtype=np.int32)
    return np.sum((factor1 * factor2), axis=-1)

**Circle LBP**
def circle_LBP (img: np.array, r: int, p:int):
  - img: An image with size of H x W.
  - r:The radius of circle, e.g., 3
  - p:The number of sampling points, e.g., 8
  h, w = img.shape
  dst = np.zeros((h, w), dtype=img.dtype)

  for i in range(r, h-r):
    for j in range(r, w-r):
      LBP_str = []

      for k in range(p):
          # **STEP 1**: sampling point
          rx = i + r*np.cos(2*np.pi*k/p)
          ry = j - r*np.sin(2*np.pi*k/p)

          # **STEP 2**: cal pix val from its surrounding four points
          x0 = int(np.floor(rx)) // top left point
          x1 = int(np.ceil(rx))  // top right point
          y0 = int(np.floor(ry)) // bottom left point
          y1 = int(np.ceil(ry))  // bottom right point

          # **STEP 3**: // final pixel value
          f00 = img[x0, y0]
          f01 = img[x0, y1]
          f10 = img[x1, y0]
          f11 = img[x1, y1]

          w1 = x1 - rx
```

```
            w2 = rx - x0
            w3 = y1 - ry
            w4 = ry - y0

            fxy = w3*(w1*f00 + w2*f10) + w4*(w1*f01+w2*f11)

            # **STEP 4**: compare tiwh neighbour point to center point
            if fxy >= img[i, j]:
                LBP_str.append(1)
            else:
                LBP_str.append(0)

        LBP_str = ''.join(str(id) for id in LBP_str)
        dst[i, j] = int(LBP_str, 2)

    return dst

  **SIFT**
  sift = cv2.SIFT_create()
  def sift_feat(img):
      kps, des =  sift.detectAndCompute(img, None)
      responses = [kp.response for kp in kps]
      order = np.argsort(responses)[::-1]
      return np.array(des[order[:30]])
```