LÝ THUYẾT

Thị giác máy tính là một lĩnh vực trí tuệ nhân tạo (AI), cho phép máy tính và hệ thống lấy thông tin có ý nghĩa từ hình ảnh kỹ thuật số, video và các đầu vào trực quan khác, và thực hiện hành động hoặc đưa ra đề xuất dựa trên thông tin đó. Nếu AI cho phép máy tính suy nghĩ, thì thị giác máy tính cho phép chúng nhìn, quan sát và hiểu. (xác định vật thể, theo dõi vật thể, đo đạc, phát hiện vật thể, phân loại vật thể).

Nhập môn thị giác máy tính là môn học thuộc lĩnh vực Thị giác máy tính. Nội dung môn này học chưa liên quan đến ngữ nghĩa của hình ảnh, chỉ học các thuật thoán ở cấp độ pixel, cụm pixel hay toàn bộ bức ảnh.
Trái với Nhập môn thị giác máy tính, **Thị giác máy tính nâng cao** là môn học liên quan đến ngữ nghĩa của hình ảnh. Nội dung môn này bao gồm lý thuyết nâng cao, các chủ đề nghiên cứu hiện tại trong thị giác máy tính với trọng tâm là các nhiệm vụ nhận dạng và học sâu, các phương pháp tiếp cận thực tế để xây dựng các hệ thống Thị giác máy tính thực sự.

Thao tác trên pixel là những thao tác có sự tính toán và tác động chỉ trên một điểm ảnh. Vd: push – wipe – uncover effect, blending effect, animation, tăng giảm độ sáng.
Thao tác trên cụm pixel là những thao tác có sự tính toán trên một vùng ảnh và chỉ có tác động trong phạm vi vùng ảnh đó. Vd: correlation, convolution, pooling, morphology.
Thao tác trên toàn bộ ảnh là những thao tác có sự tính toán và tác động trên toàn bộ tấm ảnh. Vd: seam carving, chromakey, cân bằng histogram.

Các bước thực hiện so khớp ảnh:
- B1 – Detect Feature: chọn các đặc trưng nổi bật và có tính bất biến như điểm ở góc – corner hoặc các khối tròn – blobs, bằng hàm Laplace Gaussian.
  + Tại sao ngta ưu dùng điểm ở góc hơn ở cạnh làm điểm đặc trưng: dễ dàng tìm ra được trên ảnh cần so khớp hơn so với điểm ở cảm.
  + Dùng Blob để làm đặc trưng sẽ có lợi gì so với điểm ở góc: nếu dùng điểm thì sẽ rất nhiều cách mô tả, còn blob là duy nhất, mô tả dễ dàng.
- B2 – Describe Feature: Mô tả các đặc trưng dưới dạng các vector bằng thuật toán SIFT.
- B3 – Match Feature: So khớp đặc trưng bằng thao tác Similarity (consine/ tích vô hướng) hoặc các độ đo khoảng cách (L1, L2)

Morphology là tập hợp các thao tác xử lí hình ảnh dựa trên hình dạng. Các phép toán morphology áp dụng một structure element vào input img để tạo ra output img.
- Khử nhiễu; Cô lập các phần tử riêng lẻ và kết hợp các phần tử khác nhau trong một hình ảnh; Tìm các vết lồi hoặc lõm có trong hình ảnh.

1. **Dilation:** duyệt kernel B qua hình ảnh, tính toán giá trị pixel tối đa được phủ bởi B và thay thế pixel hình ảnh ở vị trí anchor bằng giá trị tối đa đó. Kết quả của phép toán này là làm cho các vùng sáng trong hình ảnh giãn nở.
2. **Erosion:** duyệt kernel B qua hình ảnh, tính toán giá trị pixel tối thiểu được phủ bởi B và thay thế pixel hình ảnh ở vị trí anchor bằng giá trị tối thiểu đó. Kết quả của phép toán này là làm cho vùng sáng trong hình ảnh trở nên mỏng đi.
3. **Closing:** dilation → erosion. Mục đích là thu hẹp các khoảng trống trong hình ảnh mà khổng ảnh hưởng đến các đối tượng quá xa nhau.
4. **Opening:** erosion → dilation. Mục đích là giữ lại các pixel của input img mà phù hợp với structuring element. Hay nói cách khác, opening giúp mở các khoảng trống trong hình ảnh.

LẬP TRÌNH

Lấy kích thước ảnh: h, w, d = img.shape
Lấy giá tị màu ở một pixel: (B, G, R) = img[50, 50]
Lật ảnh theo chiều dọc: result = img[::-1, :]
Lật ảnh theo đường chéo chính: result = cv2.transpose(img)
Lật ảnh theo chiều ngang: result = img[:, ::-1]
Thay đổi độ sáng của ảnh: result = α*img - β //$0 < \alpha \leq 1, 0 \leq \beta \leq 255$
Thay đổi giá trị vùng ảnh (dòng a→b-1, cột c→d-1): img[a:b, c:d] = 255
Ảnh âm bản: result = 255-img
RGB→Gray: result = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
Pha trộn màu: result = cv2.addWeighted(img1, a, img2, b, 0) // b = 1-a

Push effect:
```
for D in range(0, h+1, stride):
  result[0:h-D, :, :] = img1[D:h, :, :]
  result[h-D:h, :, :] = img2[0:D, :, :]
  cv2.imshow("Push Effect", result)
  cv2.waitKey(5)
```

Uncover effect:
```
for D in range(0, w+1, stride):
  result[:, 0:w-D, :] = img1[:, D:w, :]
  result[:, w-D:w, :] = img2[:, w-D:w, :]
  cv2.imshow("Uncover Effect", result)
  cv2.waitKey(5)
```

Blending effect:
```
sliders = [img1, img2]
output = cv2.VideoWriter('video.avi', cv2.VideoWriter_fourcc(*'DIVX'), 60, (500, 500))
prev_img = slider[1]
for slider in sliders:
    for i in range(100):
        alpha = i/100
        beta = 1-alpha
        fig = cv2.addWeighted(slider, alpha, prev_img, beta, 0)
        output.write(fig)
    prev_img = slider
output.release()
```

Pooling:
```
def pooling(img, kernel_size=2, stride = 1):
    h = (img.shape[0] – kernel_size)//stride+1
    w = (img.shape[1] – kernel_size)//stride+1
    result = np.zeros(shape=(h,w))
    for i in range(0, h):
        for j in range(0, w):
            //max: np.max(), median: np.median(), average: np.mean()
            result[i, j] = np.max(img[i*stride:i*stride+kernel_size,
                                      j*stride:j*stride+kernel_size])
    return result
```

Filter thông dụng:

Lấy cạnh:
```
np.array(
  [[1, 2, 1],
   [0, 0, 0],
   [-1, -2, -1]]
)
```

Sắc nét:
```
np.array(
  [[0, -1, 0],
   [-1, 5, -1],
   [0, -1, 0]]
)
```

Làm mờ:
```
1/9 *np.array(
  [[1, 1, 1],
   [1, 1, 1],
   [1, 1, 1]]
)
```

Correlation:
```
cross_correlation(img, filter, padding=True):
    if(padding):
        top_pad = bot_pad = filter.shape[0]//2
        right_pad = left_pad = filter.shape[1]//2
        img = cv2.copyMakeBorder(
          img,
          top=top_pad,
          bottom=bot_pad,
          left=left_pad,
          right=right_pad,
          borderType=cv2.BORDER_CONSTANT,
          value=[0, 0, 0]
        )
    h = img.shape[0] – fiter.shape[0] + 1
    w = img.shape[1] – fiter.shape[1] + 1
    result = np.zeros(shape=(h, w))
    for i in range(h):
        for j in range(w):
            arr = img[i : i+filter.shape[0], j : j+filter.shape[1], :]
            result[i][j] = np.sum(arr*filter)
    return result
```

Covolution:
Tương tự correlation, chỉ đổi dòng code:
```
arr = img[i : i+filter.shape[0] : -1, j : j+filter.shape[1] : -1, :]
```

Histogram (ảnh xám):
```
def create_hist(img):
    hist = np.zeros(256,np.int32)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            hist[img[row, col]] += 1
    return hist
def display_hist(img):
    data = img.flatten()
    fig, ax = plt.subplots(figsize=(10, 7))
    ax.hist(data, bins = [i for i in range(255)])
    plt.show()
```

Cân bằng Histogram: là sự điều chỉnh histogram về trạng thái cân bằng, làm cho phân bố (distribution) giá trị pixel không bị co cụm tại một khoảng hẹp mà được "kéo dãn" ra. Gọi Z là hàm tích lũy, mức sáng i được tính là: $K(i) = \frac{Z(i)-\min(Z)}{\max(Z)-\min(Z)} * 255$

```
def equal_histogram(hist):
    Z = np.array(hist.cumsum())
    K = (Z - Z.min())/(Z.max()-Z.min())*255
    return K
```

```
def apply_ehist(img, K):
    result = img.copy()
    h, w = result.shape
    for i in range(h):
        for j in range(w):
            result[i][j] = K[result[i][j]]
```

LẬP TRÌNH

Template Matching by Cross Corelation:

```
def aspect_distance(arr1, arr2):
    mul = np.sum(arr1.astype("float")*arr2.astype("float"))
    sum1 = np.sum(arr1.astype("float")**2)
    sum2 = np.sum(arr2.astype("float")**2)
    return mul/((sum1*sum2)**0.5)    #0<result<1
def cross_correlation(img, filter, padding=True):
    // đã trình bày trước đó, thay dòng code:
    result[i][j] =  aspect_distance(arr, filter)
def template_matching(img, pattern, threshold=0.8):
    after_cor = cross_correlation(img, pattern, padding=True)
    result = img.copy()
    h, w, _ = pattern.shape
    for i in range(0, after_cor.shape[0]):
        for j in range(0, after_cor.shape[1]):
            if(after_cor[i][j] > threshold):
                l = j - w//2, t = i - h//2, r = j + w//2, b = i + h//2
                result = cv2.rectangle(result, (l, t), (r, b), (255, 0, 0), 2)
  return result
```

Chroma key:

```
def get_threshold(self, rois, k=2):
    b = [], g = [], r = []
    for roi in rois:
       b += roi[:, :, 0].flatten().tolist() // tương tự với g, r
    b = np.array(b) // tương tự với g, r
    mean = np.array([np.mean(b), np.mean(g), np.mean(r)])
    var = np.array([np.var(b), np.var(g), np.var(r)])
    sigma = np.sqrt(var)
    threshold = []
    threshold.append(mean - k*sigma)
    threshold.append(mean + k*sigma)
    return threshold
def apply_background(self, background = None):
    new_img = self.img.copy()
    rois = self.get_rois()
    threshold = self.get_threshold(rois)
    mask = cv2.inRange(new_img, threshold[0], threshold[1])
    if background is None:
        new_img[mask!=0] = np.array([0, 0, 0])
    else:
        h, w, _ = new_img.shape
        new_background = cv2.resize(background, (w, h))
        for i in range(h):
            for j in range(w):
                if(mask[i, j] != 0):
                    new_img[i, j, :] = new_background[i, j, :]
    return new_img.copy()
```

ỨNG DỤNG

Bài toán đếm tế bào (input ảnh xám):

B1: Đưa ảnh về dạng nhị phân ($i \in \{0, 255\}$), đối $tượng = 255$

```
auto_thresholding bằng otsu's method:
    def otsu(img):
        hist = img.flatten()
        thresholds = list(set(hist))
        intra_class_var = sys.maxsize
        threshold = 0
        for i in range(1, len(thresholds)):
            class1 = hist[np.where(hist<thresholds[i])]
            class2 = hist[np.where(hist>=thresholds[i])]
            var1 = np.var(class1)
            var2 = np.var(class2)
            if intra_class_var > len(class1)*var1+len(class2)*var2
                intra_class_var = len(class1)*var1+len(class2)*var2
                threshold = thresholds[i]
    binary_img = img.copy()
    binary_img[np.where(img>=threshold)] = 0
    binary_img[np.where(img<threshold)] = 255
    return binary_img
```

B2: Áp dụng erode, dilate trong cv2 để tách các tế bào rời nhau.

B3: Áp dụng findContours và drawContours để xác định số lượng thành phần liên thông và trực quan kết quả.

Bài toán đếm hàng hóa trên hệ thống băng chuyền:

B1: Đọc video: cap = cv2.VideoCapture(video_path)

Cách đọc mỗi frame:

```
while(True):
    ret, frame = cap.read()
    if not ret:
        break
```

B2: Tách nền cho các đối tượng đang di chuyển (gói hàng): abs(curr_frame – prev_frame) ta sẽ lấy được cạnh của đối tượng có sự di chuyển:

```
diff = cv2.absdiff(prev_frame, curr_frame)
```

Bước này cần trả về danh sách diff giữa các frame → diffs

B3: Trên mỗi diff, áp dụng template matching với pattern là ảnh gói hàng, nơi nào đạt ngưỡng thì ghi nhận tọa độ (gọi là package_cor).

B4: Lần lượt đánh index cho các gói hàng trên diff[0].

B5: Từ diff[1]: Tính ngưỡng IoU đối với mỗi package_cor của cur_diff bắt cặp với tất cả package_cor của prev_diff:

gói hàng nào của cur_diff tồn tại bắt cặp có giá trị IoU ≥ threshold → đánh index giống với gói hàng mà nó được so.

gói hàng nào của cur_diff tất cả các bắt cặp đều < threshold → đánh index mới

B6: Tổng số gói hàng đã chạy trên băng ghi hình bằng index lớn nhất.

THƯ VIỆN

CV2: import cv2

```
cv2.cvtColor(img, cv2.COLOR_BRG2GRAY)
cv2.erode(img, kernel, iter=1) → 1 img (mặc định anchor là tâm kernel)
cv2.dilate(img, kernel, iter=1) → 1 img (mặc định anchor là tâm kernel)
cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_
    NONE) → contours, hierarchy (contours là tập hợp các bộ tọa độ
    (x,y) là đường bao cho mỗi TPLT, số lượng TPLT = len(contours))
cv2.drawContours(img, contours, -1,(0, 255, 0),3) → vẽ trực tiếp lên img
cv2.inRange(img,upper_bound, lower_bound) → 1 img, ngưỡng là [R,
    G, B] (trả về 255 nếu pixel nằm trong ngưỡng, 0 nếu ngược lại).
cv2.filter2d(img, -1, kernel) → 1 img
cv2.fillConvexPoly(mask, npar, 255), vẽ trực tiếp lên mask, npar là
    np.arr các điểm
cv2.Canny(img, low_thres, high_thres) → 1 gray img (phát hiện cạnh)
Tạo video:
  fourcc = cv2.VideoWriter_fourcc(*'MP42')
  video = cv2.VideoWriter('video.avi', fourcc, float(24), (width, height))
  video.write(frame)
  video.release()
```

NUMPY: import numpy as np

```
Phép toán ma trận: *, +, -, np.abs(arr), np.sqrt(arr),  → 1 ma trận
Phân phối: np.var(arr), np.mean(arr), np.median(arr), np.sum(arr),
np.min(arr), np.max(arr), np.argmin(arr), np.argmax(arr) → 1 số nguyên
Làm phẳng: arr.flatten()
Gán trên vùng: arr[mask>threshold] = 255  // hạn chế: gt gán cố định
Gán trên vùng: result = np.where((arr>threshold), 255, arr2)
Chuyển về mảng: arr.tolist()  // để ghép mảng lại (np ko làm đc)
Trả về kết quả phép & trên tất cả phần tử: np.all()
Trả về kết quả phép || trên tất cả phần tử: np.any()
```

IMAGEIO: import imageio

```
Convert từ đối tượng np sang imageio: imageio.core.util.Array(img)
Tạo gif: imageio.mimsave(path, frames)
```

IMUTILS: import imutils

```
Xoay ảnh: imutils.rotate_bound(img, angle=90)
```