CS4243
AY23/24 Sem 2
github.com/jasonqiu212

05. Segmentation

- Goal: Separate image into coherent regions
- Idea: **Clustering** - Group similar data points together
- Challenges: What makes 2 points same/different? Choice of features (e.g. Color, Intensity, Position), Which clustering algorithm?
- **k-Means Clustering** - Iteratively re-assign points to nearest cluster center
  1. Given K, randomly initialize the cluster centers $c_1, \dots, c_K$
  2. For each point p_i, find the closest c_j and put p_i into cluster j
  3. Set c_j to be mean of points in cluster j
  4. Repeat, if c_j have changed up to some threshold
  - Pros: Simple, Converges to local min.
  - Cons: Setting K, Sensitive to initial centers (Since k-means converges to local min.), Sensitive to outliers (Can add more clusters), Assumes spherical clusters (Fix with mean-shift)
- **Simple Linear Iterative Clustering (SLIC) Superpixels**
  - **Superpixel** - Group of pixels that share common traits
    - Application: Inputs to other CV algo. since more compact representation with perceptual meaning
  - Num. of pixels: n_{tp}; Target num. of superpixels: n_{sp}
  - Initial width of each superpixel: $s = \sqrt{\frac{n_{tp}}{n_{sp}}}$
  - Features: $z = [r, g, b, x, y]$
  - Color distance: $d_c = ||\langle r_j, g_j, b_j \rangle - \langle r_i, g_i, b_i \rangle||$
  - Spatial distance: $d_s = ||\langle x_j, y_j \rangle - \langle x_i, y_i \rangle||$
  - Scaling factors: d_{cm} and $d_{sm} = s$ set as max. expected values of d_c and d_s respectively
  - $D = \sqrt{(\frac{d_c}{d_{cm}})^2 + (\frac{d_s}{d_{sm}})^2} = \sqrt{d_c^2 + (\frac{d_s}{s})^2 c^2}$
  1. Split img. into grid of size $s \times s$. Set cluster centers as lowest gradient position in 3×3 neighborhood from superpixel center to speed up convergence since initialize on value common to surrounding.
  2. For each cluster center, check distance to all pixels within $2s \times 2s$ neighborhood. Assign pixels to closest checked center.
  3. Update cluster centers using mean and repeat if not converged (Same as k-Means)
  4. Optional: Replace superpixel region by average value to create stained glass effect
- Modification of k-Means: Not random initialization, Compute pixel's distance only to closest set of cluster centers
- Can enforce connectivity and use other features too
- **Mean-Shift Clustering** - Find local density maxima in feature space
  - **Attraction basin** - Region in feature space for which all trajectories of centroids lead to same mode
  - **Cluster** - All data points in attraction basin of a mode
  1. For each data point:
     (a) Define window around and get centroid
     (b) Shift window to centroid
     (c) Repeat until window centroid stops moving
  - Segmentation with Mean Shift: Do mean shift and merge pixels in same attraction basin
  - Choosing window size: Trial and error, Sample points and use avg. dist. to knn. (Num. of neighbors needs to be large enough to ensure increase in density)
    - Larger window size → Fewer clusters
  - Pros: No assumptions on cluster shape, 1 parameter, Finds variable num. of modes (vs. specified k in k-Means), Robust to outliers
  - Cons: Choosing h, Slow, Scales poorly with feature space dimension
  - Optimizations:
    - After each run of mean shift, assign all points within radius r of end point to same cluster
    - Assign all points within radius $c < r$ of search path to mode → More aggressive, less confident

06. Texture

- **Texture** - Pattern with repeating elements
- **Filter Bank** - Measures variety of structures in local neighborhood and generates multi-dimensional features
  - Goal: How to represent texture?
  - Idea: Apply filters with small windows to generate statistics that summarize local patterns. Dist. in feature space bet. windows → Pixel's texture similarity
  - d filters → d-dimensional feature vector
  - Choosing window size: Try many sizes and look for one where statistic does not change much
  - Choose filters in different scales and orientations (to solve window size problem)
  - **Gabor Filter** - Represent filter banks mathematically by combining sinusoids with exp. (Gaussian) envelope
- **Texton** - Characterizes texture by replacing each pixel with integer representing **texture type**
  1. Apply filter bank to training image
  2. Cluster in feature space and store cluster centers (Texton dictionary)
  3. For test image, filter image with same filter bank to get feature vector for each pixel. Assign each pixel to nearest cluster. Cluster ID = Texton ID.
  4. For a given region, compute **texton histograms**
  - Classification: Given new img., compare hist. to other trng. samples and assign to label with most similarity
  - Segmentation: Use texton histograms as a feature
- **Perceived Boundaries** - Segmentation by human
  - Idea: Texture gradients indicate boundaries well
  1. For each pixel, consider a disk that is split into 2 halves with some orientation
  2. Measure texture diff. bet. 2 halves via texton hist.
  3. Try all orientations. Orientation with high difference suggests boundaries.

07. Keypoints

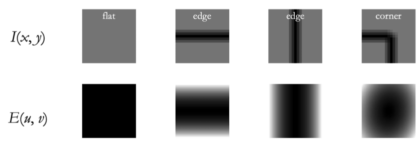
- Motivation: How to stitch 2 images (e.g. Panaroma)?
  1. Keypoints: Find locations
  2. Descriptors: Rep. surrounding regions with math
  3. Homography: Do the matching
- Good keypoints are **repeatable** and **distinct**
- **Harris Corner Detection**
  - Significance: Corners have big changes in all directions when shifting window
  - Given window W shifted by offset (u, v):

$$E(U, v) = \sum_{(x,y) \in W} (I(x+u, y+v) - I(x, y))^2$$

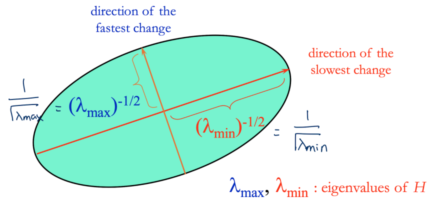
  - Assuming only small shifts (for Taylor Series Exp.):

$$E(u, v) = Au^2 + 2Buv + Cv^2 = \begin{vmatrix} u & v \end{vmatrix} \begin{vmatrix} A & B \\ B & C \end{vmatrix} \begin{vmatrix} u \\ v \end{vmatrix}$$

  - $A = \sum_{(x,y) \in W} I_x^2 \quad B = \sum_{(x,y) \in W} I_x I_y$
  - $C = \sum_{(x,y) \in W} I_y^2$
  - **2nd Moment Matrix (H)** - Middle matrix



  - $E = k$ visualized as ellipse, where H controls shape
    - Eigenvectors of H → Axes orientation
    - Eigenvalues of H → Axes length



  - **Eigenvectors** of A are vectors $\mathbf{x}$ that: $A\mathbf{x} = \lambda\mathbf{x}$
  - **Eigenvalue** (λ) corresponds to $\mathbf{x}$: $\det(A - \lambda I) = 0$
  - Since $A = H$ is 2×2: $\lambda_\pm = \frac{1}{2}((h_{11} + h_{22}) \pm \sqrt{2h_{12}h_{21} + (h_{11} - h_{22})^2})$
  - After getting λs, find $\mathbf{x}$: $(A - \lambda I)\mathbf{x} = 0$
  - Both λ_{max} and λ_{min} are large → Corner
  - 'Cornerness' Score: $R = \min(\lambda_1, \lambda_2)$ (But getting λ is slow)
  - **Harris Operator**: $R = \det(H) - \kappa(\text{trace}(H))^2$
    - $\det(H) = AC - B^2 = \lambda_1 \lambda_2$
    - $\text{trace}(H) = A + C = \lambda_1 + \lambda_2$
    - $R > 0$ → Corner, $R < 0$ → Edge, $R \approx 0$ → Flat
    1. Compute gradient for each point in image
    2. Compute H matrix for each image window and get 'correctness' score
    3. Find points with window where $R >$ threshold
    4. Take points of local maxima
  - **Non-Max. Suppression** - Iteratively search for max. values, then zero everything in surrounding window
    - Window size important
    - Adaptive: To prevent uneven distri. of keypoints in areas of higher contrast, pick corners which are both local max. and whose response is greater than all neighboring local max.
  - In practice, $H = \sum_{(x,y) \in W} w_{x,y} \begin{vmatrix} A & B \\ B & C \end{vmatrix}$ (Gaussian)
- **Harris Corner Invariances**
  - Purpose: If img. transf., how repeatable is detection?
  - **Equivariance** - Image transformed, and detection location undergoes similar transformation
  - **Invariance** - Image tranf., but no detection score change
  - Translation: Equivariant and invariant
  - Rotation: Equivariant and invariant
  - Photometric transformation (Assume $I' = aI + b$): Invariant to $b \neq 0$, but not invariant to $a \neq 1$
  - Scaling: Not equivariant (i.e. Img. scaled up)
    - Scale of window can determine if location is keypoint → Need to scale up window by image scale
    - **Auto. Scale Selection** - When looking for keypoints, try window sizes and find scale that gives local max.

- **Laplacian of Gaussian** - Alternative keypoint detector which detects 'blobs' and is scale-sensitive
  - $\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$
  - Idea: Convolution with LoG has highest response when signal has same scale as Gaussian. Built-in scale sensitivity by varying scale σ.
  - Implementation: Fix window and kernel size; rescale img. with Gaussian blurring and downsampling

08. Descriptors

- Goal: Get feature vector surrounding each keypoint

09. Homography

10. Optical Flow

11. Tracking

12. Deep Learning