CSE 659A: Advances in Computer Vision

Spring 2019: T-R: 2:30-4pm @ Cupples II/230

Instructor: Ayan Chakrabarti (ayan@wustl.edu).

http://www.cse.wustl.edu/~ayan/courses/cse659a/

Feb 19, 2019
REMINDER: Project 1 Proposals due tonight 11:59pm.
Homework 2 Review due Thursday.
So far for geometric reasoning, we have tried finding correspondences between images of the same object ... robust to viewpoint change deformations, lighting changes, etc.
One way of thinking of semantic tasks is that it is finding correspondences between an image, and a canonical image or template of that object (e.g., a car).

So there is work in vision on finding interest point "detectors" and region "descriptors".

Detectors: Give us a sparse sense of discriminative regions to match across images / templates. Regions must be unique enough to have only a few possible matches.

Descriptors: Feature representation of image regions that can be matched while being robust to viewpoint and lighting changes.
We are given an image. We don't want to compute a dense flow or disparity map. 
- Instead, we find "roughly" match the image to another.

- Applications might be object recognition, or say we want to fit a homography or fundamental matrix between two images.
If we want a rough match, we don't need to find matches for all pixels. Instead, we want to find a subset of points or regions to match.

Desiderata for these regions:
- There are few of them in each image (so that we can consider all pairs).
- This subset should be interesting or unique: so that for a reference point, there won't be too many other good candidates.
We only need a few matches: so let's try to match those where the matching problem is easy.

Superpixels?

Reduces the number of matches, but nothing discriminative about superpixels. Also, segmentation may not be consistent across views.
Enter SIFT.

- Interesting Regions = Blobs
SIFT DETECTOR

Basic Idea: Convolve image with a "blob" filter, and find pixels where the response is high, and a local maximum or minimum in space and scale.

T. Lindeberg, Feature detection with automatic scale selection, IJCV 30(2), pp 77-116, 1998
SIFT DETECTOR

Source: Lana Lazebnik

Basic Idea: Convolve image with a "blob" filter, and find pixels where the response is high, and a local maximum or minimum in space and scale.
- Blob filter we use is a Laplacian, or a double derivative of a Gaussian.

\[ \nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \]

Where \( g(x, y) \) is a Gaussian function.
• Blob filter we use is a Laplacian, or a double derivative of a Gaussian.

\[ \nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \n\]

• Remember we used first derivatives of Gaussians to detect edges.
• Second derivatives peak at blobs = combinations of separated edges in opposite directions.

But only if we select the Gaussian width \( \sigma \) to match the size of the blob.
SIFT DETECTOR

Incorrect $\sigma$ will give us pairs of responses at edges. Correct $\sigma$ will give us a single maximal response at center of blob.

Solution: Instead of convolving image with one Laplacian filter, convolve it with a bunch of Laplacians with different scale / variance $\sigma$ of the underlying Gaussian.

Pick the $(x, y, \sigma)$ where the magnitude of the response is higher than in the 3D neighborhood:

- In $x$, $y$, and in $\sigma$. 

Source: Lana Lazebnik
• Pick the \((x, y, \sigma)\) where the magnitude of the response is higher than in the 3D neighborhood:
  - In \(x, y\), and in \(\sigma\).
Problem: Pure Laplacian response goes down as you increase $\sigma$.

- Lindberg paper: shows that you must normalize by multiplying $\sigma^2$ to your filter.

$$\sigma^2 \nabla^2 g_\sigma = \sigma^2 \left( \frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2} \right)$$
Convolving with a bunch of Laplacians is expensive.

\[ L = \sigma^2 \left( G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right) \]

(Laplacian)

\[ DoG = G(x, y, k\sigma) - G(x, y, \sigma) \]

(Difference of Gaussians)

Can approximate Laplacian as a difference of Gaussians with different \( \sigma \).
• Another problem: Laplacian fires not just on blobs, but also on edges.
• This would be fine, except that the "center" of an edge is poorly "along" the edge.
• Solution: Make sure that there's curvature in orthogonal directions.

\[ H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \]

Compute the Hessian matrix like we did for corner detection (and in optical flow).

• Make sure that the ratio of first to second eigen-value isn't too large. (Can be done based on trace and determinant, instead of explicit eigendecomposition).

So overall criteria: Maxima in scale and space, not an edge (based on eigenvalue ratio), and also that magnitude of response is above some threshold.
• Now that we have a region, we need some way to "describe them".

• Get a feature vector $F$ for the region so that low distance between two regions in feature space means regions match.

• Needs to be invariant to lighting, rotation, scale.
• Needs to be invariant to lighting, rotation, scale.
• Scale: Just use the detected scale to resize the region to a canonical sized patch.

If our canonical size is $P \times P$, take the region of size $\sigma P \times \sigma P$, and resize it to $P \times P$.

• Lighting / intensity invariance: Compute a normalized histogram over gradient directions within the patch.
• Bin angles from 0 to 360 into a fixed number of bins.
• Each bin has contributions weighted by gradient magnitude and distance from center of patch.
- Bin angles from 0 to 360 into a fixed number of bins.
- Each bin has contributions weighted by gradient magnitude and distance from center of patch.
- Can select the peak of histogram as 'dominant' orientation, and shift the histogram to make it 0 degrees. (Corresponds to rotating the patch).
SIFT DESCRIPTOR

Source: Lana Lazebnik

Full version: Create separate histograms for different sub-regions, and concatenate them all together.
• We have "uniform" scale invariance: gives you invariance if you zoom in or out of the image.
• But what about 3D rotations? These will correspond to affine transformations.

• Resizing by detected scale won't normalize for skew.
What we want is to represent "scale" as an ellipse: showing the relative scaling in two identified orthogonal directions (i.e., rotation and eccentricity).
• After detection and uniform scale normalization, consider second moment matrix of detected blob.

\[ M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \]

Recall:
\[ \begin{bmatrix} u \\ v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const} \]

• Apply affine transform to map this to a circle, and then compute features.


SIFT APPLICATIONS

- Use in HW paper 2 to do initial sparse matching to find fundamental matrices between two frames.
- Use to compute homographies between multiple images. This is how multiple images are stitched into panoramas and photo-spheres in your phones.

Source: DP-Review
SIFT APPLICATIONS

- Use in HW paper 2 to do initial sparse matching to find fundamental matrices between two frames.
- Use to compute homographies between multiple images. This is how multiple images are stitched into panoramas and photo-spheres in your phones.
- Basis of the PhotoTourism method that applies this to many pairs of images in a photo collection or album.

- But SIFT was also used in pre-deep learning solutions for semantic vision tasks.
CONTENT-BASED IMAGE RETRIEVAL

- Basically, Image Search. Given a query image, find matching images from a dataset.

- Want to match the same scene, be invariant to viewpoint changes, lighting, etc.
• Find a similarity metric based on detected SIFT (and other detector region) features.

Basic idea: map text search algorithms to the image search task.

- Step 1: map detected SIFT vectors to words.
- Step 2: Documents (images) are bags of words. Match them by relative occurrence of words.

Step 1: map detected SIFT vectors to words.

- Find detected regions in a large number of images.
- For each image $D$ with $R$ detected regions, you have a set $\{f_{rd}\}$ of feature vectors of detected regions.
- But these are continuous. We want to build a fixed "vocabulary" of "words".
- Use K-Means to group them into $K$ clusters.
- The cluster IDs are your words. So now you have a set $\{w_{rd}\}$ where each $w_{rd} \in \{1, 2, \ldots K\}$. Each image is a "document" with a set of words $I = \{w_r\}$.

Some Implementation Details:

1. Don't use euclidean distance on $f$ itself, first normalize as $f' = \Sigma_f^{-1/2}(f - \mu_f)$, where $\mu_f, \Sigma_f$ are computed over the entire dataset.
2. Use multiple detectors, and cluster each of them separately into separate words. So $I = \{w_r\} \cup \{w'_r\}$, $w_r \in \{1, \ldots K_1\}, w'_r \in \{K_1 + 1, \ldots K\}$.
3. Drop all clusters (and their associated regions in an image) that are too large or too small---i.e., the very common and very rare words.

So now, we have a representation of an image as a (variable length) set of "words".

Idea from text-match: TF-IDF representation

- Term Frequency, Inverse-Document Frequency: Define a vector \( t \in \mathbb{R}^K \) of length = size of your vocabulary:

\[
    t_k = \frac{n_{kd}}{n_d} \log \frac{N}{n_k}
\]

- \( n_{kd} \): Number of times word of type \( k \) occurs in this specific image \( d \).
- \( n_d \): Number of words (i.e., detected regions) in this specific image (\( \sum_k n_{kd} \)).
- \( n_k \): Number of times word of type \( k \) occurred in the entire dataset or corpus.
- \( N \): Number of total words in the entire corpus.

Frequency of different words in a document \( \times \) log inverse frequency in corpus.

- Compare two images as the normalized dot-product between two tf-idf vectors.

- Could find similar frames from the same movie (many results on Groundhog Day and Run Lola Run though).

- But in general, image search is a hard problem. Building a "vocabulary" of all possible photographs in the world is hard!
DENSE SIFT-LIKE HUMAN DETECTION


Problem 1: Humans may appear at different scales in images.

Search over all windows of that size at multiple scales.

Binary Classification Task:
 Window at correct location at correct scale is 1
 Everything else is 0.

Fix a canonical detection window size
DENSE SIFT-LIKE HUMAN DETECTION


At a given scale:

Binary Classification Task:
Window at correct location at correct scale is 1
Everything else is 0.

Fix a canonical detection window size

For any given window, they learn a linear classifier on SIFT features.

For any given window, they learn a linear classifier on SIFT features.

Actually, they use a similar representation called HoG (Histogram of Gradients).

But they only use the descriptor. Instead of detecting regions, they compute descriptors for all overlapping patches of a certain size (which is relative to the scale).
DENSE SIFT-LIKE HUMAN DETECTION


\[ F(x,y,:) = \text{HoG}(\text{patch centered at } x,y) \]

Classifier: \[ \phi^T F > 0 \]

Learn \( \phi \) on a training set of positive and negative windows.
Linear classifier so easy to learn (learn as an SVM with hinge-loss).

For any given window, they learn a linear classifier on SIFT features.
Actually, they use a similar representation called HoG (Histogram of Gradients).

But they only use the descriptor. Instead of detecting regions, they compute descriptors for all overlapping patches of a certain size (which is relative to the scale).
DENSE SIFT-LIKE HUMAN DETECTION

- Fixed window size (say $W_1 \times W_2$).
- $F[x, y, :]$ computed as HoG features within each patch in window. Classifier is $\Phi^T F$.
- Implemented in practice as $F[x, y, :]$ as full image-sized feature tensor, i.e., for all overlapping windows (at that scale).
- Get score matrix for all windows by convolution: $\text{Conv}(F, \Phi)$, where $\Phi$ is a $W_1 \times W_2 \times H \times 1$ kernel, with $H$ the size of the HoG feature vector.
- Think of Dalal & Triggs as learning a single linear convolutional layer for the final output, which takes a handcrafted feature representation of the image as input.
- Was easy to learn and didn't require much data.
- But modern network architectures can be thought of as trying to mimic these operations. Where we have a bunch of conv layers to "do the work" of extracting a better feature representation than HoG.
- In fact, Dalal & Triggs also used other steps: eg., "local contrastive normalization". Features of a patch where normalized by mean / std of feature vectors in neighboring patches.
- The NIPS 2012 ImageNet network (Alexnet) had local contrastive normalization layers inspired by this.