Reminder: Homework 2 Review due Thursday.

Last Time

- Talked about how "correspondences" relate to semantics:
  - Correspondences b/w different views gives us geometry.
  - Correspondences b/w different examples of the same object can help with recognition.
- Consider settings where we don't want "dense" correspondences, but reliably match a sparse set of points/regions between two images.
- Talk about region "detectors": that find "unique" regions that are "easy" to match across images.
Enter SIFT.

- Interesting Regions = Blobs
Enter SIFT.

- Interesting Regions = Blobs
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
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.
SIFT DETECTOR

Source: Lana Lazebnik

- Blob filter we use is a Laplacian, or a double derivative of a Gaussian.

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

- 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.
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 $\sigma$. 
SIFT DETECTOR

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.

\[
\sigma^2 \nabla^2 g_\sigma = \sigma^2 \left( \frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2} \right)
\]
SIFT DETECTOR

Source: Lana Lazebnik

Original signal

Unnormalized Laplacian response

Scale-normalized Laplacian response

maximum
- Convolving with a bunch of Laplacians is expensive.

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

\[
DoG = G(x, y, k\sigma) - G(x, y, \sigma) \\
\text{(Difference of Gaussians)}
\]

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 (note these are second derivatives, not moments).

- 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.
SIFT DESCRIPTOR

Source: Lana Lazebnik

- 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).
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).
SIFT DESCRIPTOR

Source: Lana Lazebnik

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

- 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 detected region \( r \) in a given image \( d \), 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 \( I = \{ w_r \} \cup \{ w'_{r'} \} \), where \( w_r \in \{ 1, \ldots K_1 \} \), \( w'_{r'} \in \{ K_1 + 1, \ldots K \} \).
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 total 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.
DENSE SIFT-LIKE HUMAN DETECTION


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

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.

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).

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

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.
- Modern CNN 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.
Limitations of this Approach

- Using hand-crafted SIFT features.
- Also learning a single template for the entire object.
- But an object has many parts. A linear filter says my kernel contains "expected" feature values for different parts in corresponding regions of the kernel. And I'm doing detection by correlation.
DEFORMABLE PARTS MODEL

Source: Ross Girshick


Describe an object as a "graph" over parts.
DEFORMABLE PARTS MODEL

Source: Ross Girshick


Again have a graphical model: $G = (V, E)$

- One vertex for location of every part $V = (v_1, \ldots, v_n)$
- Edges $E$ between pairs where we specify the expected range of displacements $s$ between them.
- Random variables are locations $\{p_i = [x_i, y_i]\}$ for each part in the input image.
DEFORMABLE PARTS MODEL

Source: Ross Girshick


Score for a Candidate Detection of Object:

\[
\text{Score}(p_1, \ldots p_n) = \sum_{i=1}^{n} m_i(p_i) - \sum_{(i,j) \in E} d_{ij}(p_i, p_j)
\]

- \(m_i(p_i)\): This comes from a part classifier for part type i, applied to a region around \(p_i\).
  - \(m_i(p_i) = \text{Conv}(F, \Phi_i)[p_i]\)
Score for a Candidate Detection of Object:

\[
\text{Score}(p_1, \ldots p_n) = \sum_{i=1}^{n} m_i(p_i) - \sum_{(i,j)\in E} d_{ij}(p_i, p_j)
\]

- \(d_{ij}(p_i, p_j)\): Comes from a "spring cost". Typically expressed as \(d(p_i - p_j - (p_i^0 - p_j^0))\)
  - A cost of deviation between their actual relative distance and expected relative displacement.
DEFORMABLE PARTS MODEL


\[
\text{Score}(p_1, \ldots p_n) = \sum_{i=1}^{n} m_i(p_i) - \sum_{(i,j) \in E} d_{ij}(p_i, p_j)
\]

Let's say you had learned this model already.

- At inference time, how do you find detected objects: need to score all possible assignments of \((p_1, \ldots p_n)\).
- Each \(p_i\) can be anywhere in the image ...
- Huge search space \((WH)^n\), but can solve in polynomial time if graph is a tree.
DEFORMABLE PARTS MODEL

Source: Ross Girshick


- Star Model: There is a "root" part, and you only have edges between the root part and all other parts.
DEFORMABLE PARTS MODEL

Source: Ross Girshick


- Star Model: There is a "root" part, and you only have edges between the root part and all other parts.

\[
\text{Score}(p_0, \ldots, p_n) = \sum_{i=0}^{n} m_i(p_i) - \sum_{(i,j) \in E} d_{ij}(p_i, p_j)
\]

\[
\downarrow
\]

\[
\text{Score}(p_0) = \max_{(p_1, p_2, \ldots, p_n)} m_0(p_0) + \sum_{i=1}^{n} m_i(p_i) - d_i(p_i - p_0)
\]

- So you are searching over locations for the root part, and it's score is given by maximizing over the relative locations of other parts.

- Corresponds to doing \(n\) independent maximizations over each \(p_i, i > 0\).