A (Very Short) Primer to Image Processing

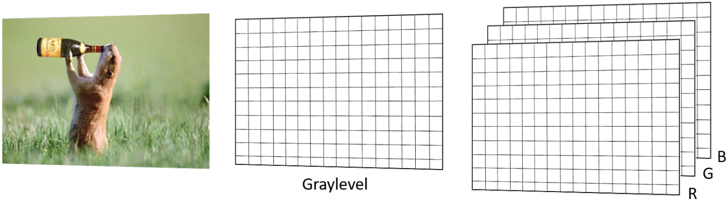
Davide Bacciu

Dipartimento di Informatica
Università di Pisa
bacciu@di.unipi.it

Intelligent Systems for Pattern Recognition (ISPR)

Image Format

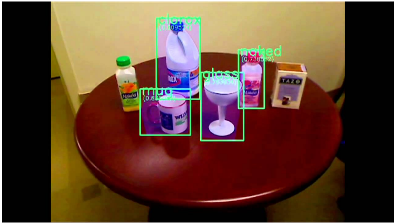
Images are matrices of pixel intensities or color values (RGB)



- Other representations exist, but not of interest for the course
- CIE-LUV is often used in image processing due to perceptual linearity
  - Image difference is more coherent

Machine Vision Applications

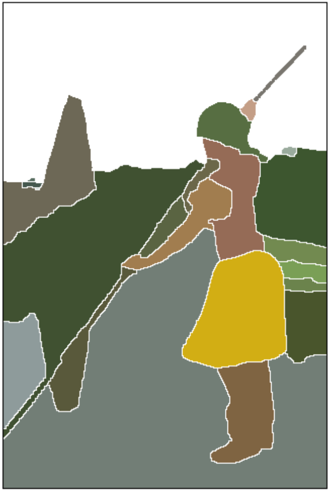
Region of interest identification



Object classification

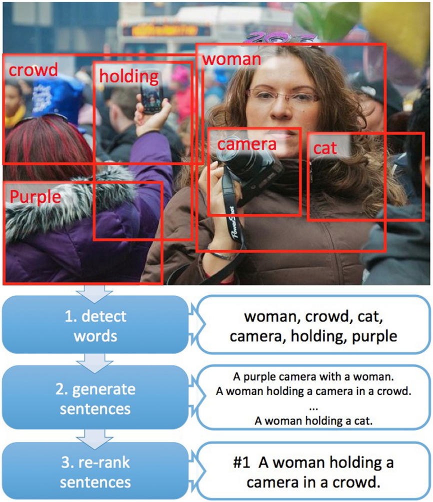
Machine Vision Applications

Image Segmentation



Semantic segmentation

Machine Vision Applications



Automated image captioning

...and much more

Key Questions?

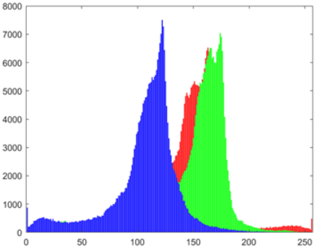
- How do we represent visual information?
  - Informative
  - Invariant to photometric and geometric transformations
  - Efficient for indexing and querying
- How do we identify informative parts?
  - Whole image? Generally not a good idea...
  - Must lead to good representations
  - Edges, blobs, segments

Image Histograms

- Represent the distribution of some visual information on the whole image
  - Color
  - Edges
  - Corners
- Color histograms are one of the earliest image descriptors
  - Count the number of pixels of a given color (normalize!)
  - Need to discretize and group the RGB colors
  - Any information concerning shapes and position is lost

Color Histograms

Images can be compared, indexed and classified based on their color histogram representation

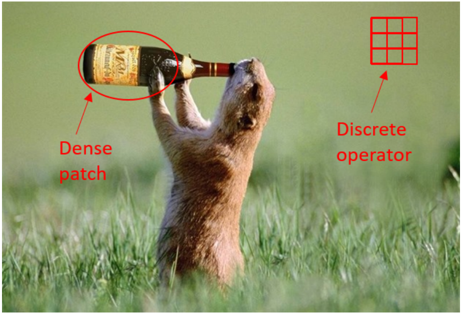



```
%Compute histogram on single
    channel
[yRed, x] = imhist(image(:,:,1));
%Display histogram
imhist(image(:,:,1));
```

```
import cv2 #OpenCV
image = cv2.imread("image.png")
# loop over the image channels
chans = cv2.split(image)
colors = ("b", "g", "r")
for (chan, color) in zip(chans, colors):
        hist = cv2.calcHist([chan], [0], None, \
            [256], [0, 256])
```

Describing Local Image Properties

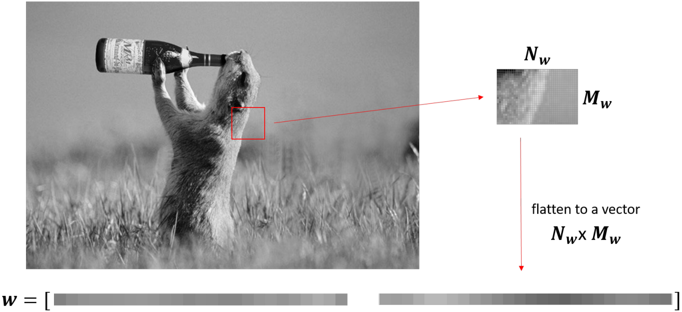
- Capturing information on image regions
- Extract multiple local descriptors
  - Different location
  - Different scale
- Several approaches, typically performing convolution between a filter and the image region



Need to identity good regions of interest (later)

Intensity Vector

The simplest form of localized descriptor



Normalize **w** to make the descriptor invariant w.r.t. affine intensity changes

$$d = \frac{\mathbf{w} - \overline{w}}{\|\mathbf{w} - \overline{w}\|}$$

- No invariance to pose, location, scale (poorly discriminative)

Distribution-based Descriptors

Represent local patches by histograms describing properties (i.e. distributions) of the pixels in the patch

- What is the simplest approach you can think of?
  - Histogram of pixel intensities on a subwindow
  - Not invariant enough
- A descriptor that is invariant to
  - Illumination (normalization)
  - Scale (captured at multiple scale)
  - Geometric transformations (rotation invariant)

Scale Invariant Feature Transform (SIFT)

1. Center the image patch on a pixel x, y of image I
2. Represent image at scale σ
   - Controls how close I look at an image

Convolve the image with a Gaussian filter with std σ

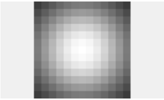
$$L_\sigma(x, y) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Gaussian Filtering of an Image

Create the Gaussian filter

```
%A gaussian filter between -6 and +6
h=13, w=13, sigma=5;
%Create a mesh of pixel points in [-6,+6]
[h1 w1]=meshgrid(-(h-1)/2:(h-1)/2, -(w-1)
    /2:(w-1)/2);
%Compute the filter
hg= exp(-(h1.^2+w1.^2)/(2*sigma^2));
%Normalize
hg = hg./sum(hg(:));
```



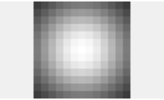
Then, convolve it with the image

Or you use library functions to do all this for you

Gaussian Filtering of an Image

Create the Gaussian filter

```
%A gaussian filter between -6 and +6
h=13, w=13, sigma=5;
%Create a mesh of pixel points in [-6,+6]
[h1 w1]=meshgrid(-(h-1)/2:(h-1)/2, -(w-1)
      /2:(w-1)/2);
%Compute the filter
hg= exp(-(h1.^2+w1.^2)/(2*sigma^2));
%Normalize
hg = hg./sum(hg(:));
```



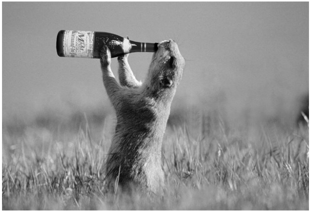
Then, convolve it with the image

Or you use library functions to do all this for you

```
Iscale = imgaussfilt(I, sigma);
```

$\sigma = 5$

$\sigma = 0.05$



Scale Invariant Feature Transform (SIFT)

1. Center the image patch on a pixel x, y of image I
2. Represent image at scale σ
3. Compute the gradient of intensity in the patch
   - Magnitude m
   - Orientation θ

Use finite differences:

$$m_\sigma(x, y) =$$

$$\sqrt{(L_\sigma(x+1, y) - L_\sigma(x-1, y))^2 + (L_\sigma(x, y+1) - L_\sigma(x, y-1))^2}$$

$$\theta_\sigma(x, y) = \tan^{-1}\left(\frac{(L_\sigma(x, y+1) - L_\sigma(x, y-1))}{(L_\sigma(x+1, y) - L_\sigma(x-1, y))}\right)$$

Gradient and Filters

A closer look at finite difference reveals

$$G_x = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * L_\sigma(x, y)$$

$$G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * L_\sigma(x, y)$$

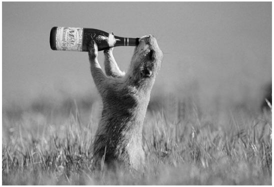
So

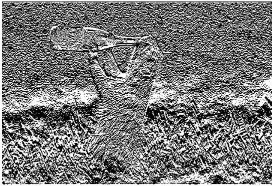
$$m_\sigma(x, y) = \sqrt{G_x^2 + G_y^2} \text{ and } \theta_\sigma(x, y) = \tan^{-1}\left(\frac{G_y}{G_x}\right) =$$

Gradient Example

```
%Compute gradient with central difference on x,y directions
[Gx, Gy] = imgradientxy(Ig,'central');
%Compute magnitude and orientation
[m, theta] = imgradient(Gx, Gy);
```

I_g m θ

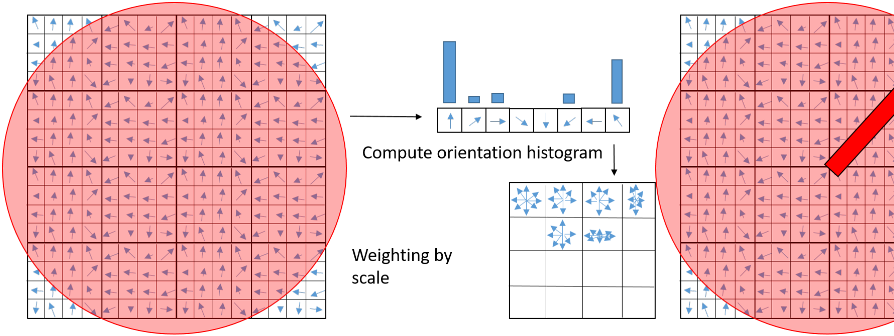




Scale Invariant Feature Transform (SIFT)

1. Center the image patch on a pixel x, y of image I
2. Represent image at scale σ
3. Compute the gradient of intensity in the patch
4. Create a gradient histogram
   - 4x4 gradient window
   - Histogram of 4x4 samples per window on 8 orientation bins
   - Gaussian weighting on center keypoint (width $= 1.5\sigma$)
   - $4 \times 4 \times 8 = 128$ descriptor size

SIFT Descriptor



- Normalize to unity for illumination invariance
- Threshold gradient magnitude to 0.2 to avoid saturation (before normalization)
- Rotate all angles by main orientation to obtain rotational invariance

SIFT Facts

- For long time the most used visual descriptor
  - HOG: Histogram of oriented gradients
  - SURF: Speeded Up Robust Features
  - ORB: an efficient alternative to SIFT or SURF
  - GLOH: Gradient location-orientation histogram
- SIFT is also a detector, although less used

SIFT in OpenCV

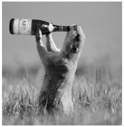
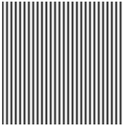
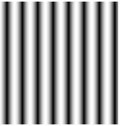
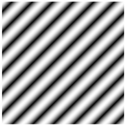
```
import cv2
... #Image Read
gray= cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
sift = cv2.xfeatures2d.SIFT_create()
#1 — Detect and then display
kp = sift.detect(gray,None)
kp,des = sift.compute(gray,kp)
#2 — Detect and display
kp,des = sift.detectAndCompute(gray,None)
```

Fourier Analysis

- Images are functions returning intensity values $I(x, y)$ on the 2D plane spanned by variables x, y
- Not surprisingly, we can define the Fourier coefficients of a 2D-DFT as

$$H(k_x, k_y) = \sum_{x=1}^{N-1} \sum_{y=1}^{M-1} I(x, y) e^{-2\pi i \left(\frac{x k_x}{N} + \frac{y k_y}{M} \right)}$$

In other words, I can write my image as sum of sine and cosine waves of varying frequency in x and y directions


$= c_1$

$+ c_2$

$+ c_3$

$+ \cdots$

The Convolution Theorem

The Fourier transform $\mathcal{F}$ of the convolution of two functions is the product of their Fourier transforms

$$\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$$

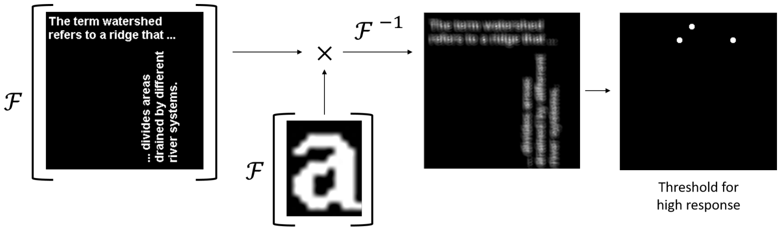
- Transforms convolutions in element-wise multiplications in Fourier domain
- Suppose we are given an image I (a function) and a filter g (a function as well)...
- ...their convolution $I * g$ can be conveniently computed as

$$I * g = (F)^{-1}(\mathcal{F}(I)\mathcal{F}(g))$$

where $(F)^{-1}$ is the inverse Fourier transform

Convolutional neural networks can be implemented efficiently in Fourier domain!

Image PR with DFT

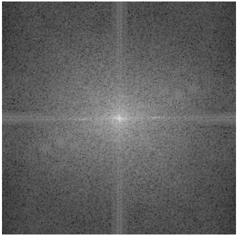


1. Make a filter out of a pattern using Fourier transform $\mathcal{F}$
2. Convolve in Fourier domain and reconstruct with $\mathcal{F}^{-1}$
3. Threshold high pixel activation to generate response mask

Practical Issues with DFT on Images

Previous example, in Matlab:

```
[N,M] = size(I);
mask = ifft2(fft2(I) .* fft2(charPat,N,M)) > threshold;
```



- The DFT is simmetric (in both directions):
  - Power spectrum is re-arranged to have the $(0, 0)$ frequency at the center of the plot
- The $(0, 0)$ frequency is the DC component
  - Its magnitude is typically out of scale w.r.t. other frequencies

$$H(0,0) = \sum_{x=1}^{N-1} \sum_{y=1}^{M-1} I(x,y)e^{0}$$

  - Use $log(abs(H_{.,.}))$ to plot the spectrum (or log-transform the image)

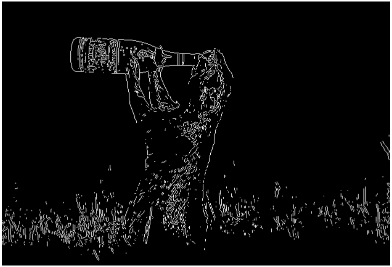
Visual Feature Detector

Repeatability

Detect the same feature in different image portions and in different images

- Photometric - Changes in brightness and luminance
- Translation - Changes in pixel location
- Rotation - Changes to absolute or relative angle of keypoint
- Scaling - Image resizing or changes in camera zoom
- Affine Transformations - Non-isotrophic changes

Edge Detection



Edges and Gradients

- Image gradient (graylevel)

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

  direction of change of intensity
- Edges are pixel regions where...
  - Intensity gradient changes abruptly
- The return of finite difference methods

$$G_x = \frac{\partial I}{\partial x} \approx I(x+1, y) - I(x-1, y)$$

$$G_y = \frac{\partial I}{\partial y} \approx I(x, y+1) - I(x, y-1)$$

G_x

$$\begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix}$$

G_y

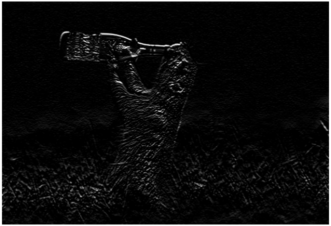
$$\begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Prewitt operators

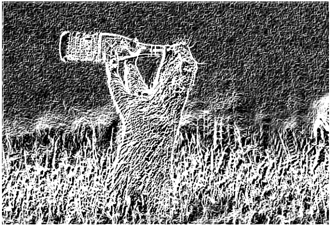
Convolving Gradient Operators

Image

G_x



Magnitude

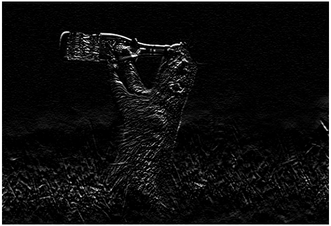


G_y

Sobel Operator

An additional level of smoothing of the central difference

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



In Code

Matlab

```
%Create an horizontal (x) Prewitt filter
h = fspecial('prewitt'); %Try also 'sobel'
%Convolve it to the image Ig
imH = imfilter(Ig,h,'replicate');
%Transpose filter for the y-derivative
imV = imfilter(Ig,h','replicate');
%Magnitude
M = uint8(sqrt(double((imHor.^2) + (imVer.^2))));
%Then plot...
imtool(imH); %etc...
```

Python

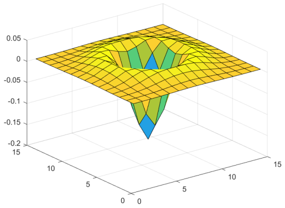
```
#prewitt masks
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])

#convolving filters
img_prewittx = cv2.filter2D(img_gray, -1, kernelx)
img_prewitty = cv2.filter2D(img_gray, -1, kernely)


#sobel (CV_8U is the output data type, ksize is the kernel size)
img_sobelx = cv2.Sobel(img_gray,cv2.CV_8U,1,0,ksize=3)
img_sobely = cv2.Sobel(img_gray,cv2.CV_8U,0,1,ksize=3)
```

Blob Detection

- Blobs are connected pixels regions with little gradient variability
- Laplacian of Gaussian (LoG) $g_\sigma(x, y)$ has maximum response when centered on a circle of radius $\sqrt{2}\sigma$



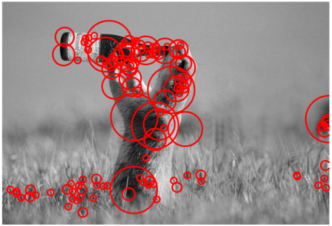
$$\nabla^2 g_\sigma(x, y) = \frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2}$$

Typically using a scale normalized response

$$\nabla^2_{norm} g_\sigma(x, y) = \sigma^2 \left(\frac{\partial^2 g_\sigma}{\partial x^2} + \frac{\partial^2 g_\sigma}{\partial y^2} \right)$$

LoG Blob Detection

1. Convolve image with a LoG filter at different scales $\sigma = k\sigma_0$ by varying k
2. Find maxima of squared LoG response
   1. Find maxima on space-scale
   2. Find maxima between scale
   3. Threshold

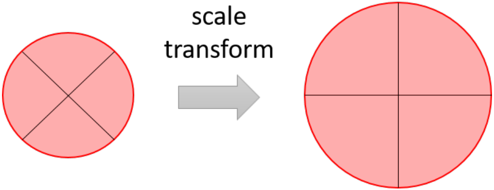


The LoG filter can be approximated as a Difference of Gaussians (DoG) for efficiency

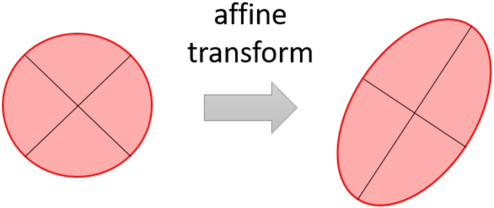
$$g_{k\sigma_0}(x, y) - g_{\sigma_0}(x, y) \approx (k-1)\sigma_0^2 \nabla^2 g_{(k-1)\sigma_0}$$

Affine Detectors

- Laplacian-based detectors are invariant to scale thanks to the maximization in scale-space



- Still not invariant to affine transformations



Maximally Stable Extremal Regions (MSER)

- Extract covariant regions (blobs) that are stable connected components of intensity sets of the image
- Key idea is to take blobs (Extremal Regions) which are nearly the same through a wide range of intensity thresholds
- The blobs are generated (locally) by binarizing the image over a large number of thresholds
  - Invariance to affine transformation of image intensities
  - Stability (they are stable on multiple thresholds)
  - Multi-scale (connected components are identified by intensity stability not by scale)
  - Sensitive to local lighting effects, shadows, etc..
- You can then fit an ellipse enclosing the stable region

Intuition on the MSER Algorithm

Generate frames from the image by thesholding it on all graylevels

- Capture those regions that from a small seed of pixel grow to a stably connected region
- Stability is assessed by looking at derivatives of region masks in time (most stable $\Rightarrow$ minima of connected region variation)

MSER in Code

Again, in OpenCV

```
import cv2
...
#Load the mser detector from OpenCV
mser = cv2.MSER_create()
regions = mser.detectRegions(img, None)
#Create a convexhull enclosing stable regions
hulls = [cv2.convexHull(p.reshape(-1, 1, 2)) for p in regions]
#Draw detected regions on image copy
vis = img.copy()
cv2.polylines(vis, hulls, 1, (0, 255, 0))
cv2.imshow('img', vis)
```

Matlab

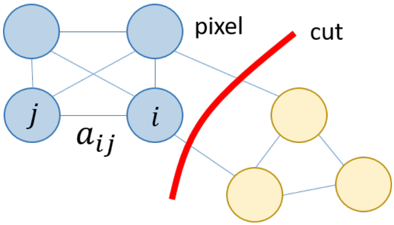
```
%Run MSER and returns regions
regions = detectMSERFeatures(Ig);
figure; imshow(Ig); %plot image
hold on;
plot(regions); %overlap regions
%Alternatively can plot actual regions
plot(regions,'showPixelList',true,'
    showEllipses',false);
```

Image Segmentation

The process of partitioning an image into set of homogeneous pixels, hoping to match object or their subparts

- A naive approach? Apply k-means to pixels color (typically L*a*b) hoping to cluster together regions
- A slightly less naive approach? Apply k-means to pixels color and (x, y) position hoping to enforce some level of spatial information in clusters

Normalized Cuts (Ncut)



- Node = pixel
- a_{ij} = affinity between pixels (at a certain scale σ)

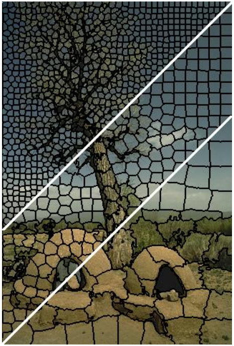
- A cut of G is the set of edges such whose removal makes G a disconnected graph
- Breaking graph into pieces by cutting edges of low affinity
- Normalized cut problem
  - NP-hard
  - Approximate solution as an eigenvalue problem

Code: `https://www.cis.upenn.edu/~jshi/software/`

Pixel Issue

Pixels in image are a lot!

- Ncut can take ages to complete
- Likewise many other advanced segmentation algorithms



- Efficiency trick ⇒ Superpixels
  - Group together similar pixels
  - Cheap, local oversegmentation
  - Important that superpixels do not cross boundaries
- Now apply segmentation/fusion algorithms to superpixels: Ncut, Markov Random Fields, etc.

Code: `https://ivrl.epfl.ch/research/superpixels`

Take Home Messages

- Image processing is very much about convolutions
  - Linear masks to perform gradient operations
  - Gaussian functions to apply scale changes (zooming in and out)
- Visual content can be better represented by local descriptors
  - Histograms of photo-geometric properties
  - SIFT is intensity gradient histogram
- Computational efficiency is often a driving factor
  - Convolutions in Fourier domain
  - Superpixels
  - Lightweight feature detector? Random sampling

Next Lecture

Generative and Graphical Models

- Introduction to a module of 6 lectures
- A (very quick) refresher on probabilities (from ML)
  - Probability theory
  - Conditional independence
  - Inference and learning in generative models
- Graphical models representation
- Directed graphical models
- Undirected graphical models