Lecture 2: Stat 238. Winter 2012. A.L. Yuille.

January 22, 2012

1 Lecture 2: Edge Detection and Multi-Scale

1. What is an Image? It is a set of intensity values (usually $\{0, 255\}$ for a black and white image) defined over a lattice. Each lattice element is called a pixel. The pizel's intensity is called the pixel value (each pixel will have three values for a color image). Hence the input to a computer vision system is a matrix of numbers (the pixel values).

2. Given a matrix-image, how could you segment it into two regions? There are two types of cues. There are big intensity changes – edges – at the boundary between the two regions. Also the intensity values are similar within each region (foreground and background). Assumes that images are piecewise smooth. Show a clean number-image and a noisy number-image with cross-sections.

3. Filtering images. Smoothing filters and derivative filters. Filterbanks (represent local properties). Convolution and Fourier theory.

4. Piecewise smooth images – a 1980's model of images. Motivates edge detection and region grouping (by similar intensity values). Empirical justification for this model based on statistics of natural images – weakness of this justification. Texture and image structures at multiple scales.

5. Multiscale. blurring. Aude Oliva – Chuck Close – Dali – Dennis Peli New York Times article. Fourier components – low frequency, high frequency. Blur with Gaussian – eliminates the high frequencies. Limitations of linear smoothing – destroys structure – alternatives later.

6. Statistical Edge Detection.

7. Bayes Decision Theory and Learning.

Techniques:

1. Calculus (derivatives and integrals).

2. Filter Theory. Fourier Theory.

3. The Diffusion/Heat equation. Nonlinear diffusion. Differential equations.

4. Basis functions – over-complete bases. Wavelets. Harmonic analysis.

5. Histrograms (non-parameteric probability distributions).

6. Log-likelihood ratio test, Bayes rule, Bayes Decision Theory.

7. Theory of Learning — memorization and generalization – cross-validation.

8. Mixtures of Distributions. Robust and non-robust distributions.

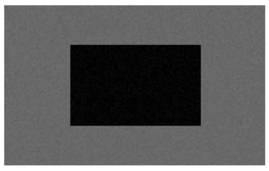
2 What is an Image?

What is the input to a computer vision system? An image is an array of intensity values: $\{I_{ij} \in [0, 255] \quad | i = 1..n, j = 1..m\}$. For example, the number matrix in Fig 1(a) describes a dark box in bright background (Fig 1(b)).

Images like (Fig 1(b)) are easy for humans to process since our brains are designed to deal with them and we can exploit all our visual knowledge. But the real input to our visual system, and to a computer vision system, is the set of numbers in Fig 1(a). The numbers represent the amount of light incident on the eye or a camera (e.g., numbers of photons).

$$\begin{bmatrix} 120 & 118 & 110 & 115 & 116 & 120 \\ 115 & 21 & 20 & 16 & 19 & 121 \\ 112 & 19 & 17 & 18 & 20 & 117 \\ 119 & 118 & 121 & 117 & 116 & 112 \end{bmatrix}$$

(a) Intensity matrix of an image



(b) Image of the matrix (150×100)

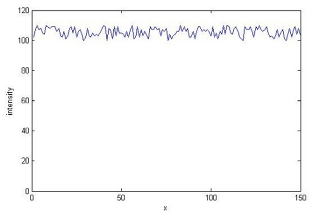
Figure 1: Image representations. Left panel: the intensity matrix is the true input to a vision system. Right panel: a conventional image which our brains are designed to process. Both representations convey exactly the same information.

Suppose you are only given the intensity values – the set of numbers $\{I(i,j)\}$ (Fig 1(a)) – but you cannot see the image. How would you start to interpret it? This is the task that a computer vision system, and biological vision systems, are faced with.

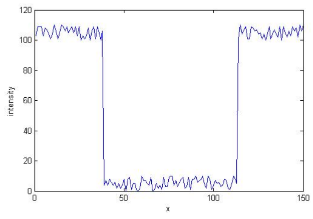
If you look at the matrix of intensities you would probably decide that it corresponds to the image of a dark square (low intensity values) surrounded by a bright background (high intensities). You are using two *cues*, *edge detection* and *region grouping*, to *segment* the images into *foreground* (center) and *background* regions.

For example, in most parts of the image the neighboring pixels typically have similar values (e.g. from 20 to 15), but sometimes there's a big jump (e.g. from 135 to 25). This can be used to detect *edges*. Also, we can group pixels with similar intensities values – which gives a complimentary way to group the pixels into two different regions. We will discuss grouping in later lectures (starting with lecture 4) and now concentrate on detecting edges.

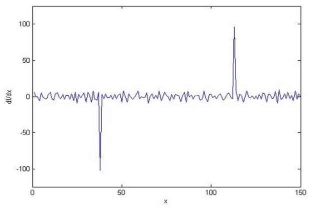
For Fig 1(b), we can differentiate the image in the x and y directions using *derivative filters* (details in the next section – e.g., derivatives, differences, filters). We plot the image intensities for one-dimensional cross-sections of the image in Fig 2 (a,b). Taking the derivative in terms of x (on cross-section with $y = 50$), we obtain the result is shown in Fig 2(c). We see that the big jumps occur at where edges are. Hence we could detect edges at places where the image gradients are very large (details in a later section).



(a) Intensity values (y=5)



(b) Intensity values (y=50)



(c) Derivative Filter response (y=50)

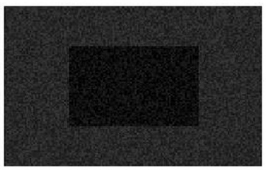
Figure 2: Edge detection by derivative filters. Left and Center panels: cross-sections of the intensity at $y = 5$ and $y = 50$ respectively. Right panel: the response of the derivative filter has large peaks at the edges (note different polarities).

However, for images which are noisier (Fig 3), there will be many places in the image where the derivatives are large. So large derivatives will not correspond to edges in the image. In this case, we need to first smooth the image before differentiating. Smoothing is performed by *filtering* with a smoothing function (see later section). It converts the image Fig 3(b) into a smoother version Fig 4)(a). The derivatives on this smoothed image are now largest at the edges and smaller elsewhere Fig 3)(b).

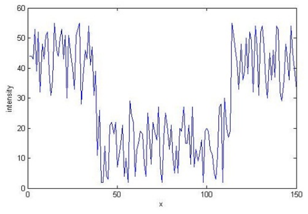
3 Linear Filtering. Convolution and Fourier Theory

This section describes introduces linear filtering. This can be used to smooth and differentiate images (gives more details for the ideas in the last section). It can also give multiscale descriptions of the image and also give dense local representations of images.

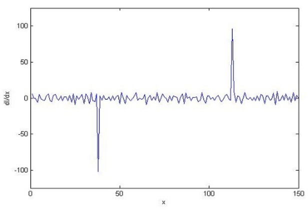
Linear filters act on an image by moving a filter G through the image and computing the integral of the product of the filter (in each position) with the images, see Fig. 5.



(a) Original image

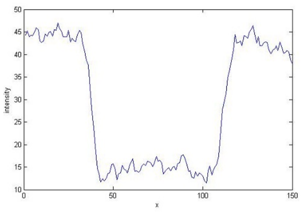


(b) Intensity values (y=50)

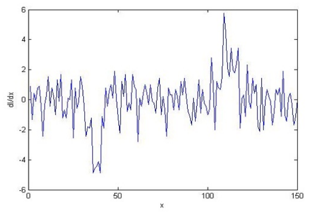


(c) Filter response (y=50)

Figure 3: A more complex image. Edge detection by derivative filters. Left panel: noisy image. Center panel: cross-section of the intensity at $y = 50$ Right panel: the response of the derivative filter has many peaks it is hard to detect the real edges (NOTE: check figure (c).



(a) Intensity values after smoothing (y=50)



(b) Filter response (y=50)

Figure 4: Edge detection by smoothed derivative filters. Left panel: the image after applying a smoothing filter. Right panel: the derivative of the smoothed images has its biggest responses at the edges.

$$F * I(\vec{x}) = \int\int dudv F(x-u, y-v:\sigma) I(u,v). \quad (1)$$

If F is a smoothing filter, see Fig. 5, it will smooth the image by performing local weighted averaging. (Note: smoothing is used in Statistics and Machine learning as a non-parametric estimate of a probability distribution. In this case the input is $I(x) = (1/N)\sum_{i=1}^{N} \delta(x - x_i)$, where $\{x_i : i = 1, ..., N\}$ are the positions of the samples and δ is the Dirac delta function. Convolving $I(x)$ with a smoothing filter F gives output $G * I(x) = (1/N)\sum_i F(x - x_i)$ – a standard non-parametric way to estimate a distribution from samples).

A particularly important smoothing filter is the Gaussian $G_\sigma(x, y) = \frac{1}{2\pi\sigma^2}\exp\{-(x^2+y^2)/2\sigma^2\}$. The standard deviation σ of the Gaussian specifies the amount of smoothing – small σ means little smoothing and large σ is a lot of smoothing. We can smooth the image at different values of σ to get representations of the image at multiple scales.

$$G_\sigma * I(\vec{x}) = \int\int dudv G(x-u, y-v:\sigma) I(u,v), \quad (2)$$

where σ is the standard deviation of the Gaussian (the larger σ the more the smoothing).

Discrete and Continuous images. Images have discrete values $I(x, y) \in \{0, 255\}$ where $x \in \{1, n\}$ $y \in \{1, m\}$ are also discrete (i.e. the pixels). But it is often convenient to think of images as being

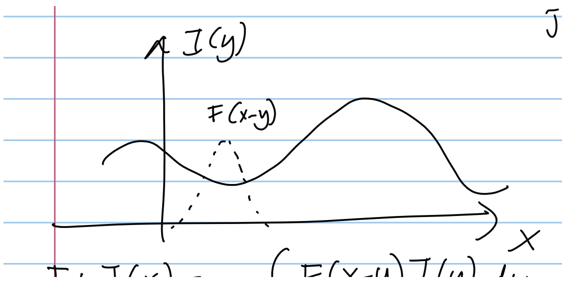


Figure 5: The image I is filtered by a function F. For each point x in the image we center the filter at x, multiply by the image, and integrate. In this case F is performing a local weighted average of the image.

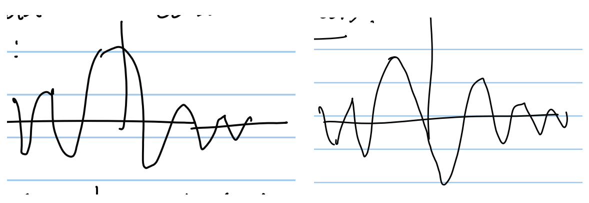


Figure 6: Left Panel: cosine Gabor. Right panel: sinusoid Gabor.

continuous differentiable functions $I(x, y)$ where (x, y) take continuous values in $D \subset R^2$. This allows us to take derivatives and use mathematical results from harmonic analysis and other disciplines. The relations between continuous and discrete images is complicated. We can think of a continuous image as being the *continuum limit* of discrete images as the distance between the pixels tends to zero. We can think of the discrete image as being sampled from a continuous image – and use Fourier analysis (e.g., Nyquist's theorem) to quantify how much information is lost. More practically, we can replace differential operators dI/dx by a difference operator $I(x + 1) - I(x)$ (recall that $dI/dx = lim_{\delta \mapsto 0} \frac{I(x+\delta)-I(x)}{\delta}$. This is called finite differences (see Wikipedia). A better approximation is $(1/2)I(x + 1) - (1/2)I(x + 1)$. A discrete second order derivative is $I(x) - (1/2)I(x + 1) - (1/2)I(x + 1)$.

For discrete images, we perform linear filtering as follows:

$$\phi * I(\vec{x}) = \int\int dudv\phi(x-u, y-v)I(u, v). \tag{3}$$

Derivative Filters: differentiating an image is also a filtering operation. Smoothing an image by a filter F and then differentiating it d/dx correspond to filtering the image by a derivative filter dF/dx. (Easy to check). Differentiating the image, without smoothing, is the same as differentiating an image which has been smoothed by a delta function – i.e. filtering the image by $d/dx\delta(x)$, where $\delta(.)$ is the Dirac delta function.

We can take as many derivatives as we like – e.g., d^2/dx^2, d^7/dx^7 – although more smoothing is usually required to smooth out the images and make high order derivatives well behaved.

For discrete images, a filter $F(x, y)$ is a derivative filter if it sums to zero – $\sum_{x,y} F(x, y) = 0$. It can be checked that this happens for the simple filter $I(x+1)-I(x)$ and $I(x)-(1/2)I(x+1)-(1/2)I(x-1)$.

Filterbanks. Filterbanks consists of several filters. They give local descriptions of images. I.e. you can describe a pixel by the the values of the filters evaluated at that point. It is a richer description that the pixel value. Many pixels in an image have the same values – but fewer have the same filterbank values (why filterbanks? – with bag of words – see below – these are one way to model textons)

Examples of filterbanks consist of derivatives of Gaussians (at different values of σ – i.e. different scales). The derivatives are in different directions and can include first, second, and higher order derivatives.

Other filterbanks are based on Gabor filters. These are sinusoids multiplied by Gaussians. They obey certain optimizality criterion in terms of their localization in the space and frequency domain (based on least square measure of locality). They are of form:

$$G_{abor}(x) = \exp\{i\vec{\omega} \cdot \vec{x}\} \exp\{-(1/2)x^T \Sigma^{-1}\vec{x}\}. \tag{4}$$

The Gabor filter is complex and can be decomposed into a cosine Gabor (the real part) and a sine Gabor (the imaginary part) ($\exp\{i\theta\} = \cos\theta + i\sin\theta$), see Fig (6). *Energy filters* are defined by summing the squares of the cosine and sine parts. (The receptive fields of some neurons, simple cells, in area V1 or the visual cortex can be approximated by Sine and Cosine Gabors and it has been argued that complex cells can be modeled as energy filters – but some neuroscientists point out that many complex cells do not have input from simple cells and also the distinction between simple and complex cells may be based on a false dichotomy and instead there are a family of cells with a range of different properties).

Wavelets (cite Wikipedia) give a way to define other filterbanks. See the Fourier series and basis function section for a discussion of these and other filters (and overcompleteness).

4 The Gaussian and the Heat/Diffusion Equation

The Gaussian filter is related to the heat/diffusion equation for the temperature T:

$$\frac{\partial}{\partial t}T(x, y, t) = \frac{\partial^2}{\partial^2 x}T(x, y, t) + \frac{\partial^2}{\partial^2 y}T(x, y, t) \tag{5}$$

The solution is

$$T(x, y, t) = G_t * T(x, y, 0) \tag{6}$$

where G_t is a Gaussian with covariance $\sigma^2 = t/2$ and where $T(x, y, 0$ is the initial conditions. It can be shown that the heat equations rapidly smooths out the input so that it tends to a spatially constant temperature – by proving that quantities which measure the smoothness of the temperature – like $\int |\nabla T_\sigma|^2 dxdy$ – decreases monotonically as σ increases.

The Gaussian has many special properties. For example, it implies that the number of edges in an image decreases monotonically with t (equivalently with σ) (Babaud et al, Yuille and Poggio). Moreover, the Gaussian is the only filter for which this is true. This is important for scale-space representations where an image is represented by its edges at multiple scales (Koenderink, Babaud et al.).

But a problem with the Gaussian is that it blurs out edges too much which also causes the positions of edges at large σ to be strongly influenced by the positions of other edges. This is highly undesirable. It motivated Perona and Malik to advocate using a non-linear diffusion equation which reduces the flow of heat at places where the intensity gradient is high (i.e. edges) and so which tends to maintain edges and not to move them. The non-linear equation they proposed in mathematically unstable (see Mumford and Shiota) but their discretized version is stable. There has been an extensive literature on related non-linear differential equations for processing images.

The relationship between the Gaussian and the diffusion equation can be exploited to demonstrate the effect of smoothing an image (see Fourier domain section). Multiscale Images:.

5 Fourier Theory and Expansions in terms of Basis Functions

Filtering is the *convolution* of the filter F with the image I. Fourier theory allows us to analyze images in their frequency domain by representing the image in terms of a sum (or integral) of sinusoids:

$$I(x, y) = \sum_{n,m} a_{nm} \exp\{inx/(2\pi)\} \exp\{imy/(2\pi)\} \tag{7}$$

where the coefficients $a_{n,m}$ are given by:

$$a_{nm} = (1/2\pi)^2 \sum_{x,y} I(x, y) \exp\{inx/(2\pi)\} \exp\{imy/(2\pi)\}. \tag{8}$$

Then the effect of smoothing by a Gaussian can be found (using the heat equation) by

$$I(x, y) = \sum_{n,m} a_{nm} \exp\{inx/(2\pi)\} \exp\{imy/(2\pi) \exp\{-(n^2 + m^2)/(2\pi)^2 t\}\} \tag{9}$$

Hence the high frequencies get removed very quickly (exponentially fast) leaving only the low frequencies.

We can apply Fourier transforms to obtain $\hat{\phi}(\omega_x, \omega_y) = \hat{\phi}(\omega_x, \omega_y)\hat{I}(\omega_x, \omega_y)$, where (ω_x, ω_y) is frequency.

For discrete images $I(\vec{x}) : x = 1, ..., 1024\ y = 1, ..., 1024$ and a discrete filter $\phi(u, v)$, we define a linear filter to be:

$$\phi * I(\vec{x}) = \sum_u \sum_v \phi(x - u, y - v) I(u, v). \tag{10}$$

The simplest filters are derivatives –e.g. d/dx and d/dy. These can be converted into discrete filters such as $1, -1$ (in one dimension). Better derivative approximations can be performed (see standard textbooks).

The fourier expansion is just one type of expansion in terms of basis functions. Many alternative basis functions for expansion are possible (e.g., Haar basis functions). Fourier is used to compress images – e.g., the standard jpeg compression – obtained by dividing the image into many small regions (of the same size) and expressing the image in each region as a truncated Fourier expansion.

An alternative is to expand an image in terms of an overcomplete set of basis functions – like wavelets – see Wikipedia.

6 Multiscale

Show some illusions. From $http://www.michaelbach.de/ot/fcs_mosaic/$.

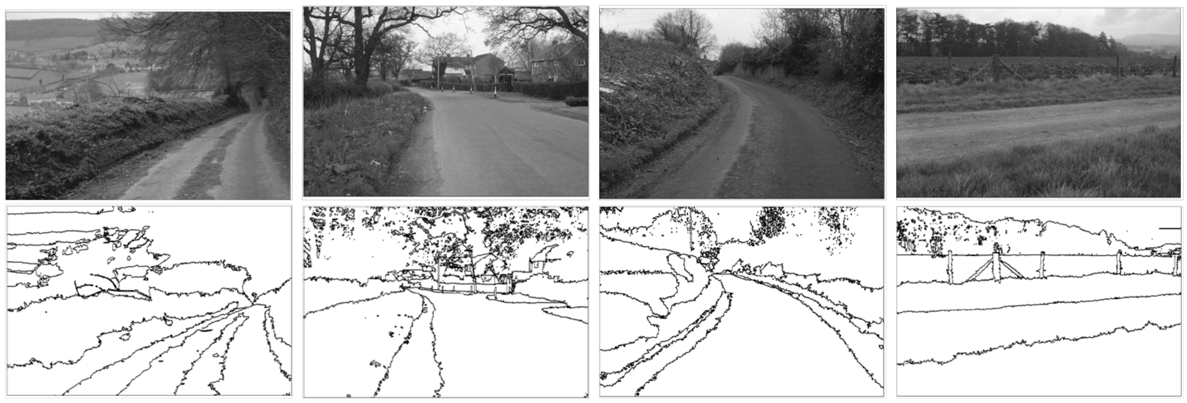
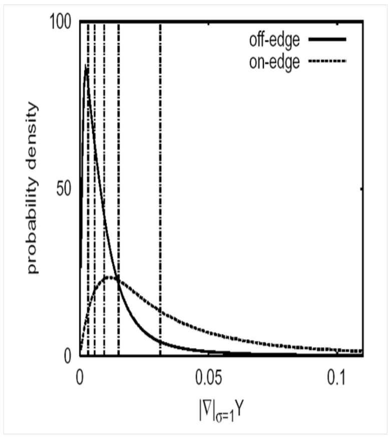


Figure 7: Upper Panels: the data images. Lower Panels: the groundtruth edge maps



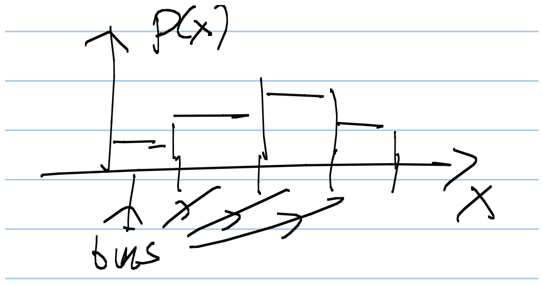


Figure 8: Left Panel: The distribution $P(.|on)$ and $P(.|off)$ represented as histograms. Observe that $logP(.|on)/P(.|off)$ is monotonically increasing as a function of the image gradient. Right Panel: a Histogram. (Add $\log P(.|on)/P(.|off)$ figure

7 Statistical Edge Detection

One of the first things to try is to detect places in the image where the intensity changes rapidly. These are *edges*. They typically occur at the boundaries of objects or at discontinuities in texture.

The most straightforward way to design an edge detector is to calculate the intensity gradient $\vec{\nabla} I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$. Label a point $\vec{x}$ as an edge if $|\vec{\nabla} I(\vec{x})| > T$, where T is a threshold. (The derivatives are approximated by the differences on the image lattice). This will give a very crude edge detector. More generally, we can define the edge detector by a filter (linear or non-linear) $\vec{\phi} * I(\vec{x})$.

How to formulate this statistically in the spirit of this course? Get a dataset of images and label the edges, see figure (7). Then learn probability distributions $P(\phi * I(\vec{x})|\vec{x} \text{ onedge})$ and $P(\phi * I(\vec{x})|\vec{x} \text{ offedge})$, see figures (8). These distributions can be represented non-parametrically (e.g., histograms) or by parameterized models (e.g., Gaussians). (These distributions should be learnt on the training dataset and tested on the test dataset and perform cross-validation to ensure generalization).

Formulate edge detection as a log-likelihood ratio test. Label a point $\vec{x}$ as an edge if $\log \frac{P(\phi * I(\vec{x})|\vec{x} \text{ onedge})}{P(\phi * I(\vec{x})|\vec{x} \text{ offedge})} > T$, where ϕ is the edge detector filter and T is a threshold. You can plot the Receiver Operating Characteristic (ROC) curve of true positives as a function of false positives (each value of T determines a point on this curve), see figure (9).

Does this improve over simply thresholding the filter response – i.e. $\phi * I(\vec{x}) > T$, for some T. The answer is often no. The reason is that $\log \frac{P(\phi * I(\vec{x})|\vec{x} \text{ onedge})}{P(\phi * I(\vec{x})|\vec{x} \text{ offedge})}$ is typically a monotonic function of $\phi * I(\vec{x})$ because $P(\phi * I(\vec{x})|\vec{x} \text{ offedge})$ is usually peaked at 0 (the derivatives of an image are usually small at most places in the image) and then gradually decreases while, by contrast, $P(\phi * I(\vec{x})|\vec{x} \text{ onedge})$ is typically small at 0, then increases for larger $\phi * I(\vec{x})$ and then decreases again – see figure (8). This monotonic relationship means it does not matter if the threshold is placed on the filter response or on the log-likelihood. So there is no advantage in using the statistical approach.

But the situation changes if you combine two, or more, different edge detectors to obtain a vector-

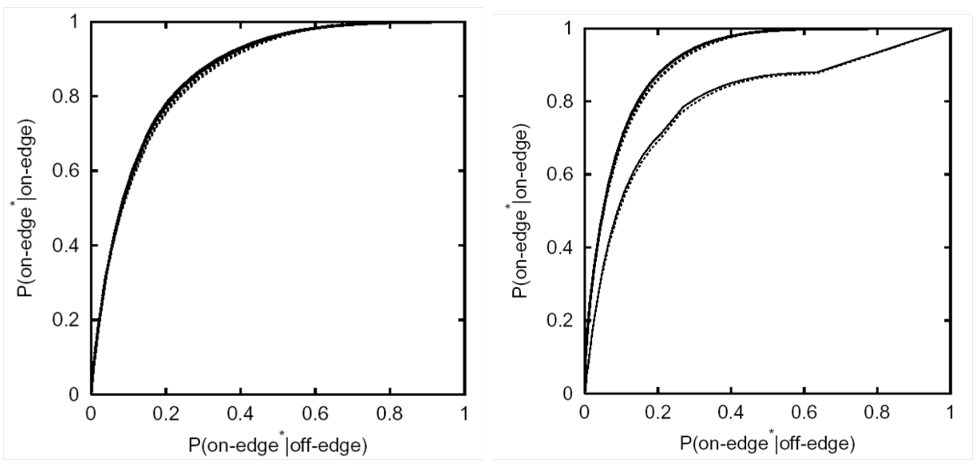


Figure 9: ROC curves. Left Panel: generalization – the ROC curves for the test and training datasets are very similar. Right Panel: failure to generalize – the ROC curves for the test and training datasets differ.

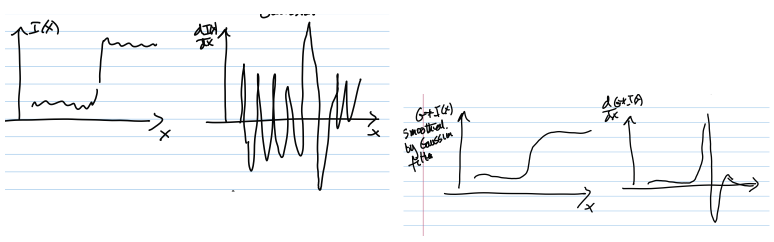


Figure 10: Derivatives of Images at different scales. Left Panel: the derivatives will be large at many places in a noisy image (even though there is only one large step edge). Right Panel: smoothing the image with a Gaussian and then differentiating will make the derivatives much smaller in most of the image but still keep a big response at the step edge (but weaker than before).

valued edge detector $\vec{\phi} * I(\vec{x})$. The statistical approach gives a natural way to combine these edge detectors. Label $\vec{x}$ as an edge if $\log \frac{P(\vec{\phi}*I(\vec{x})|\vec{x} \text{ onedge})}{P(\vec{\phi}*I(\vec{x})|\vec{x} \text{ offedge})} > T$. The results of this are superior to putting thresholds on the individual edge detectors, as can be seen from the ROC curves and other performance measures – see figure (11).

Where do these other edge detectors come from? There are a whole set of different edge detectors proposed in the computer vision literature. In particular, you can detect edges at different scales in the image.

Smoothing will also degrade large intensity gradients, which are more likely to be edges, but to a lesser extent – see figure (10). We can apply an edge detector $\phi(.)$ to the smoothed image to obtain a set of new edge detectors $\phi G_\sigma * I(\vec{x})$ by varying σ, see figure (10). We can combine these detectors to give a vector-valued detector – e.g. $\vec{\phi} = (\phi, \phi G_\sigma)$ – and learn the distributions $P(\vec{\phi} * I(\vec{x})|W(\vec{x}))$. *Important point* – there is a limit to how many filter you can combine without running out of training data. If you represent distributions by histograms, then the amount of data required scales like exponentially with the number of filters (quadratically if you use Gaussian distributions).

The performance of edge detectors can be evaluated by ROC curves or by measures of the difference between the distributions $P(.|on), P(.|off)$. The Chernoff information as a measure of the different between the distributions $P(.|on)$ and $P(.|off)$. The larger the Chernoff, the more the distributions differ. The Chernoff information $C(p,q) = -\min_{0 \leq \lambda \leq 1} \log\{\sum_y p^\lambda(y) q^{1-\lambda}(y)$. Chernoff is $(\mu_1 - \mu_2)/(8\sigma^2)$ if $p(.), q(.)$ are Gaussian distributions with identical variance σ^2 and means μ_1, μ_2. The use of Chernoff is motivated by the fact that the error rate in labeling N samples $y_1, ..., y_N$ as either all from $p(.)$, or all from $q(.)$, behaves as $\exp\{-NC(p,q)\}$ for large N. The Chernoff information also can be used to bound the Bayes risk for classifying a single sample y as being from $p(.)$ or $q(.)$. Figure (11) shows how combining different filters can lead to better edge detectors.

These statistical edge-detectors are very successful when tested on large datasets. There are improvements to the methods described here – you can design special features (Berkeley – see later), you can

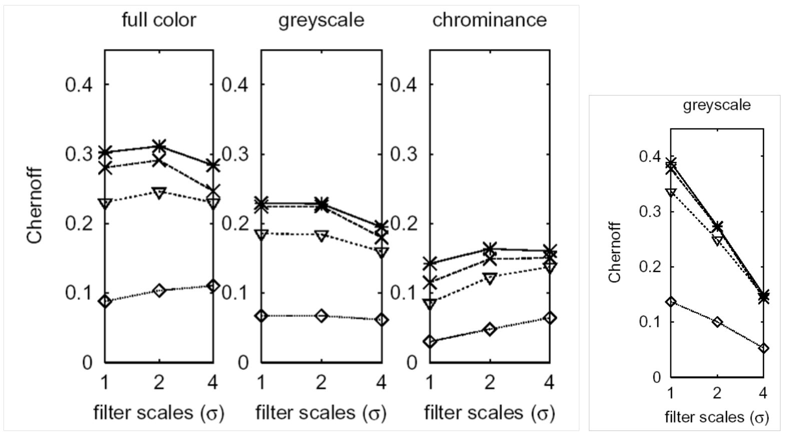


Figure 11: Evaluation of different filters and combinations of filters. Stars for the joint distribution of (N_1, N_2), crosses for N_1, triangle for $|\vec{\nabla}|$, diamonds for ∇^2. N_1, N_2 are the first and second eigenvalue of the matrix-valued operator $G * (\vec{G} * I)(\vec{G} * I)^T$, where G is a Gaussian and ^{T} denotes vector transpose.

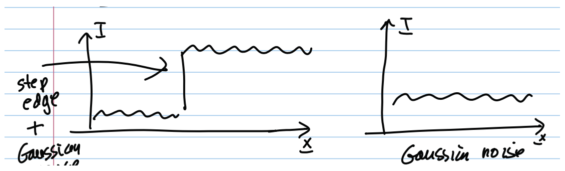


Figure 12: The assumptions made by the Canny edge detector. Left Panel: edges are step edges corrupted by additive Gaussian noise. Right Panel: non-edges are Gaussian noise.

use alternative methods for combining different edge detectors (e.g., AdaBoost). These will be described later in the course.

Now we compare statistical edge detectors to the default Canny edge detector (Canny 1983). This formulates edge detection as distinguishing between a step-edge with additive Gaussian noise and pure Gaussian noise, see figure (12) (this is an oversimplification). These can be thought of as generative models for edges and non-edges. How well do they correspond to real images? It turns out that step edges are a reasonable first order approximation to real edges (although there are a variety of other edge profiles). But non-edges are often not well described by pure Gaussian noise. It is a reasonable approximation for indoor images without texture, but fails in outdoor scenes because of vegetation (we say more about texture later in the course). Hence the performance of statistical edge detectors are significantly better than Canny on challenging outdoor images, see figure (13).

7.1 Alternative Statistical Methods

A subsequent method for statistical edge detection (Malik et al) uses a slightly different approach. They use a filterbank of Gabor filters at different scales. They apply this filterbank to all the image data and then perform a clustering algorithm to obtain a *dictionary of words* – the centers of the clusters. Then the response of the filterbank at each pixel is the word that it corresponds to.

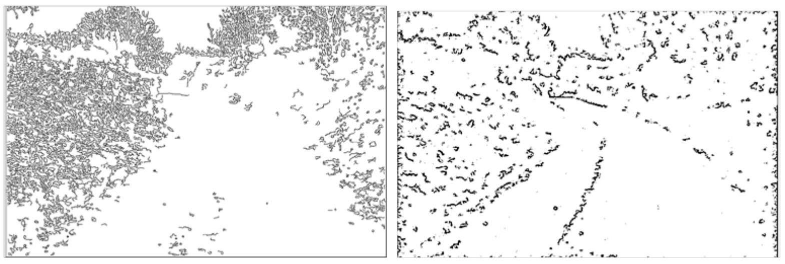


Figure 13: Comparison of Canny and Statistical Edge Detection. Left Panel: Canny edge. Right Panel: Statistical Edge Detector. Observe that Canny has bigger response on the texture in the background but also fails to detect the sides of the road.

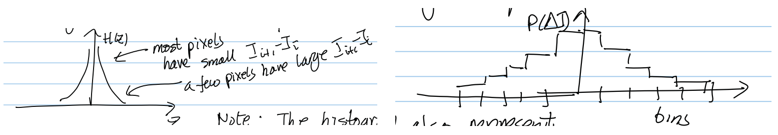


Figure 14: The histogram of the first derivative takes the following form.

At each pixel they search for edges at a range of different orientations. They select a small region of the image based on the pixel they want to classify as edge or non-edge and divide it into two regions by a line at each orientation. Then then compute the historgams of the words (of the filterbank) on each side of the line. They use a chi-squared test as a measure of similarity between the histograms. This is thresholded to give a get of edges. Next they perform a grouping of these edges – using a spectral technique (see lecture 5 or 6??) – to group edges which are supported by other edges (e.g., a prior assumption about the spatial structure of edges). The result of this edge grouping is then combined with the original edge map (how/ – weighted averaging??) to yield the edge map.

8 Are Images piecewise smooth?

Several influencial image models from the 1980's assumed that images were piecewise smooth (Geman and Geman, Blake and Zissrmann, Mumford and Shah, Osher et al.). We will discuss these models in later lectures (4 and 5??). But how good as these assumptions?

To explore this, we can differentiate an image $I(x, y)$ – to obtain dI/dx – and compute its histogram. This has been done by many people (reference). The form of the histogram is very similar for all images, see Fig 14

This histogram suggests that images really are piecewise smooth. The gradients are small for most image pixels, but for some pixels the gradients are very large. Similar plots also occur for almost all derivative filters (Lee and Mumford, Green). Also similar results arise from many other data sources (Green – and who else). Note that if the derivatives were normally distributed, then the plots would look like a Gaussian and the "tails" would fall off rapidly, like $\exp\{-(1/2)x^2\}$. But instead they fall off much more slowly – the Gaussian distribution is not "robust" enough to deal with this data (it under-estimates the changes of rare events – e.g., the black swans that arguably caused the recent recession/depression). But mixtures of Gaussians may be sufficient (e.g., Black!!).

These results suggest that images really are locally smooth – but that this "weak smoothness" is not just captured by the first order statistics. This is a pity since the 1980's models (which didn't have access to these statistics) modeled only very local interactions (see lecture 4). This was partly due to the difficulty of performing computation with that class of models if the interactions were more non-local.

9 Bayes Decision Theory and Learning

Note: this material was not covered in the lectures.

9.1 Bayes Decision Theory

So far, we have assumed that the goal of inference is to obtain the Maximum a Posteriori (MAP) estimate $W^* = \arg\max_W P(W|I)$. But why? Where does this come from? What are the alternatives?

The basis is Bayes Decision Theory which formulates problems as minimizing the expected loss. To make this precise, assume we have a joint distribution $P(W, I)$, a set Λ of decision rules $\alpha(.)$, and a loss function $L(\alpha(I), W)$ for making decision $\alpha(I)$ for input I when the true state is W.

Bayes Decision Theory states that you should pick the decision rule that minimize the risk which is defined to be the expected loss:

$$R(\alpha) = \sum_{I,W} L(\alpha(I), W) P(I, W), \tag{11}$$

where we replace the summations by integrals if I, W, or both are continuous valued.

$$\text{Bayes Rule } \alpha^* = \arg\min_{\alpha \in \Lambda} R(\alpha), \quad \text{Bayes Risk } R(\alpha^*) = \min_{\alpha \in \Lambda} R(\alpha). \tag{12}$$

NOTE: there are a few mathematical special cases where $R(\alpha^*) \neq \arg\min_{\alpha\in\Lambda} R(\alpha)$ but they very rarely occur outside mathematics books.

We can re-express the risk as:

$$R(\alpha) = \sum_I P(I) R(\alpha|I), \quad \text{where} R(\alpha|I) = \sum_W L(\alpha(I), W) P(W|I). \tag{13}$$

Hence minimizing the Bayes risk is equivalent to minimizing the conditional risk $R(\alpha|I)$ for each I (i.e., the Bayes decision rule for input I is independent of its decision for other inputs – alternatively, the distribution $P(I)$ has no effect on determining the Bayes rule). This gives:

$$\alpha^*(I) = \arg\min_{\alpha(I)} \sum_W L(\alpha(I), W) P(W|I), \text{ discrete } W$$

$$\alpha^*(I) = \arg\min_{\alpha(I)} \int dW L(\alpha(I), W) P(W|I), \text{ continuous } W. \tag{14}$$

The MAP estimate arises as a special case for a particular choice of loss function.

In the discrete case, set $L(\alpha(I), W) = 1 - \delta(\alpha(I), W)$, where the delta function $\delta(\alpha(I), W)$ takes value 1 if $\alpha(I) = W$ and value 0 otherwise (i.e., correct responses pay no penalty but incorrect responses are weighted the same). The Bayes rule reduces to maximizing $\sum_W \delta(\alpha(I), W) P(W|I) = P(W = \alpha(I)|I)$, which is the MAP estimate.

For certain applications – e.g., edge detection, face detection – it may be better to use a different loss function which penalizes false negatives (failure to find edges/faces) more than false positive (finding edges/faces where they do not exist). The reason is that we can use later processing (i.e., other models) to eliminate the false positives. But it is harder to resurrect the false negatives.

In the continuous case, set $L(\alpha(I), W) = -\delta(\alpha(I) - W)$, where $\delta(.)$ is the Dirac delta function (i.e., $\delta(x) = 0, \ x \neq 0$ and $\int dx \delta(x) = 1$, provided the range of integration contains the point $x = 0$. The Bayes rule reduces to maximizing $\int dW \delta(\alpha(I), W) P(W|I) = P(W = \alpha(I)|I)$ which is the MAP estimator.

Observe that the Dirac delta function is a strange choice of loss function because it pays an (infinite) penalty unless the decision is perfectly correct. There is no partial credit for making a decision $\alpha(I)$ that differs from the correct decision W by an infinitesimal amount. This is highly unrealistic. More reasonable choices of loss function are to set $L(\alpha(I), W) = -G(\alpha(I) - W : \sigma)$, where G is a Gaussian whose variance/covariance determines how much partial credit to give. In practice, the simplicity of MAP estimation – and the difficulty of determining how to assign partial credit – means that MAP estimation is often used in practice. Alternatives will be described later in the course. They include the mean estimate $\int P(W|I) W dW$ which occurs for quadratic loss function $L(\alpha(I), W) = (\alpha(I) - W)^2$. In summary, MAP estimation for continuous variables should be used with caution.

Bayes decision theory and approximate models. There is a simple but important conceptual point to be made from the Bayes Risk (which several well-known vision researchers have got wrong!). The Bayes risk is obtained by minimizing with respect to all decision rules. If we reduce the set of allowable decision rules, for example if we do edge detection with a restricted class of filters, then we have to do worse.

Bayes decision theory has some implications which may be counter-intuitive. Suppose the likelihood term $P(I|W)$ is sufficient to determine the correct W. Then the prior $P(W)$ may bias the result. The reason is that Bayes decision theory assumes that you should make the best decision *on average*, which does not correspond to making the best decision on a specific example. Bayes decision theory can be a bad guide for how to make one-time only decisions – such as buying a house, or gambling on the stock market. In such case, it may be wiser to take into account the worst-case loss rather than the average case (i.e., how much money can you afford to loss in the stock market). But the mathematics for this is much more complicated and worrying about the worst case is often paranoid.

9.2 Learning – Memorizing and Generalizing

The study of Machine Learning has taught us a lot about the differences of *generalizing* and *memorizing.*

All learning is done from a finite *training set* of data $\{(x_i, y_i) : i = 1, ..., N\}$ but we want decision rules to be valid for data that we have not seen yet. Most studies of this problem assume that the data samples are independent identically distributed (i.i.d.) from some unknown distribution $P(\vec{x})$. We design a decision rule $\alpha(.) \in \Lambda$ which maps x to y. We choose a loss function $L(\alpha(x), y)$ which is the penalty for making decision $\alpha(x)$ for data x when the true decision is y (i.e., usually $L(\alpha(x), y) = 0$ if $\alpha(x) = y$).

From the training dataset, we can learn a decision rule $\hat{\alpha}$ to make the *empirical risk* small, where the empirical risk (average loss over the training dataset) is defined by:

$$R_{emp}(\alpha) = \frac{1}{N} \sum_{i=1}^{M} L(\alpha(x_i), y_i). \tag{15}$$

But this is only *memorization* (i.e., applies only to the training data) unless $\hat{\alpha}$ also minimizes the *risk* (expected loss) of all the samples from the distribution:

$$R(\alpha) = \sum_x \sum_y P(\vec{x}) L(\alpha(x), y). \tag{16}$$

Memorization typically occurs when the amount N of training data is small and the set Λ of possible $\alpha(.)$ is large (technically the *capacity*). In this situation, it is too easy to find a decision rule that gives a good fit to the training data by chance (this is how conspiracy theories get started).

To obtain *generalization*, we require that $R(\hat{\alpha}) \approx R_{emp}(\hat{\alpha})$ (over-simplified?). How can we be sure that this is the case?

For certain classes of problem (making decisions) there is a beautiful theory due to Vapnik (see also Valiant) which shows that with probability greater than $1 - \delta$ that

$$R(\alpha) \leq R_{emp}(\alpha) + \phi(N, h, \delta). \tag{17}$$

where h is the VC dimension which is a measure of the capacity of the set of classifiers $\alpha \in \Lambda$. Provided N is much large than h and than $|\log \delta|$ then we can be sure that minimizing R_{emp} will make R small. But this theory is unfortunately mostly of conceptual use because the bounds are often not tight enough. The theory is slightly paranoid because it has to rule out the probability that a rule might classify the training data correctly because of a chance structure in the dataset (e.g. in high-dimension space d and only N datapoints it is always possible to get a rule to make any dichotomy). Note: Vapnik's results rely on the law of large numbers.

Instead, in practice, we use the idea of cross-validation (of which there are many variants). This involves a training dataset and a test dataset. The decision rule is learnt on the training dataset and evaluated on the test dataset (this can be thought of as using the test dataset to estimate the Bayes risk). This is not rigorously guaranteeing that $R(\hat{\alpha}) \approx R_{emp}(\hat{\alpha})$ (because there is always the chance that both training and test datasets are atypical of the distribution $P(\vec{x})$). But how paranoid do you want to be?