EN.520.665 Class Notes Taken in Fall 2020 by Mr. Vivek Jani, who enrolled in this class.

These lecture notes are based on papers and book chapters from the literature. These notes are being made available at no cost to students who enroll in EN.520.665.

Machine Perception

The Grand Challenge of Making a Computer See the world Like a Human?

Computer vision is an interdisciplinary field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. Applications are diverse and include, face matching (finding missing children, access control), diagnosis of movement-related disorders, assistive devices for the elderly, blind, and visually impaired, homeland security self driving cars, human computer interaction, commerce, and medical imaging. Over the decades, there have been many significant advances in computer vision:

- 60s - Blocks world, Robert's operator
- 70s - Image formation, shape from shading, structure from motion, Stereo
- 80s - Canny's edge detector, Regularization, MRFs, pyramids, object recognition, simulated annealing, SfM, ANNs
- 90s - Tracking, video stabilization, context, video understanding, particle filters, action recognition
- 00s - UAVs, facial recognition, gait analysis, ReID, action recognition
- 10s - Deep convolutional neural networks (DCNNs)

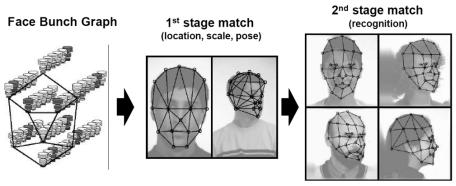


Figure: Feature based approaches of face recognition.

Consider the task of face detection. The human perception of faces is complex process. Face shape is encoded in a slightly caricatured manner, with color cues playing a crucial role, especially when shape cues are degraded. Face identity and expression might be processed by separate means. The latency of response to faces is 120 ms, suggesting a largely feed-forward computation. High-frequency information is by itself insufficient for good face recognition performance. Humans can recognize familiar faces in very low resolution images, while the ability to tolerate degradation increases with familiarity. The machine vision task of face detection up the mid 90s involved single-face segmentation, with a whole-face template and a deformable feature-based template. More recently, we are beginning to use image/appearance based methods and multi-view based detection of 3D rotated faces, utilizing techniques including optimal edge-based shape detection, adaboost methods, deformable parts models, and single stage and two-stage neural networks. Extraction of individual facial features is important for both detection and recognition and involves three approaches: (1) Generic features - fiducial points, local binary patterns, curvatures, attributes, etc. (2) Feature templates - mouth, nose, eyes, (3) Structure matching - e.g. Active Shape Model. Some challenging issues include distortion of features under large pose and "restoration" of invisible features. Examples of recognizers include principal component analysis, discriminant analysis, neural networks, graph-matching, support vector machines, and deep learning, which has recently gained much popularity. Examples of early feature based approaches (Malsburg) are shown on the right here. In 1989, LeCun et. al were the first to use backpropagation for handwritten digit recognition using backpropagation to train the weights in a convolutional neural network. Krizhevsky, Sutskever, and Hinton showed effectiveness on ImageNet (2012).

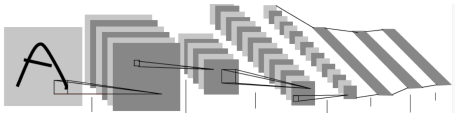


Figure: Y. LeCun, et al., Backpropagation applied to handwritten zip code recognition, Neural Computation, Vol. 1 (4), pp. 541-551, 1989.

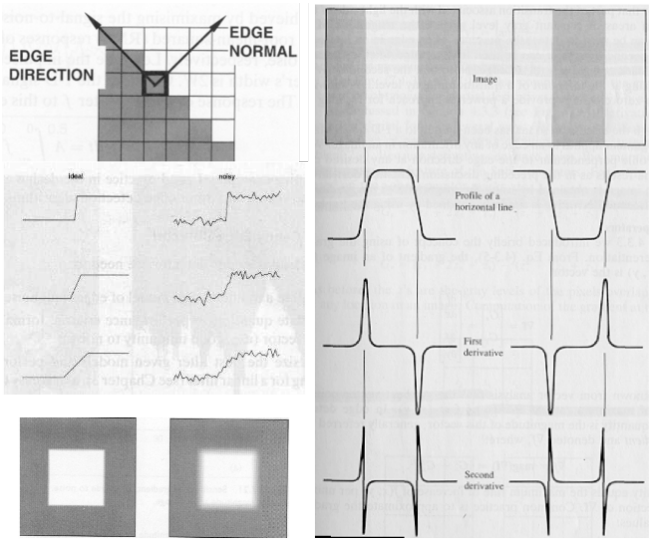
Video analysis and motion analysis from multi-camera videos has demonstrated utility in a wide variety of tasks including motion analysis in biomechanics and clinical studies, state of the art marker based motion capture, and markerless methods, which are advantageous clinically. In general computers are better than humans at performing repetitive boring tasks, low-light, might time operations (infrared, hyperspectral, synthetic aperture radar, foliage penetrating radar, ground moving target indication, radar detection), ingesting and searching over a large data base, and real-time collaborations among many agents. However, human tasks can be augmented by machine learning in many tasks, including forensic augmentation (P.J. Phillips, et al., Proc. National Academy of Sciences, May 28, 2018.).

Current research efforst revolve around mathematical models for deep learning, protecting deep learning from adversarial attacks, action detection in untrimmed videos, and deep learning in medicine. Deep learning has

already shown significant promise in a wide variety of clinical tasks, including identification of diabetic retinopathy (Gulshan, Peng, Corran, et al., JAMA, 2016.) and CT reconstruction (Lin, et al, CVPR 2019). Challenges of AI in medicine include large volumes of data with little annotation, bias, lack of explainability, high expectations, and overcoming cultural barriers.

Edge Detection - Introduction

Edges are significant local changes in intensity in an image that typically occur on the boundary between two different regions in an image. The goal of edge detection is to produce a line drawing of a scene from an image of that scene. Important features can be extracted from the edges of an image (e.g., corners, lines, curves), which can be used by higher-level computer vision algorithms (e.g. recognition). Intensity changes are caused by various physical events, including geometric events, such as object boundaries (discontinuity in depth and/or surface color and texture) and surface boundaries (discontinuity in surface orientation and/or surface color and texture), and non-geometric events, including specularity (direct reflection of light, such as a mirror), shadows (from other objects or from the same object), and inter-reflections. We describe an edge as follows:



- Edge Normal: unit vector in the direction of maximum intensity change.
- Edge Direction: unit vector to perpendicular to the edge normal.
- Edge Position or center: the image position at which the edge is located.
- Edge Strength: related to the local image contrast along the normal.

These are shown here on the right. We can model intensity changes based on their intensity profiles. Examples of such models include:

- Step edge: the image intensity abruptly changes from one value to one side of the discontinuity to a different value on the opposite side.
- Ramp edge: a step edge where the intensity change is not instantaneous but occur over a finite distance.
- Ridge edge: the image intensity abruptly changes value but then returns to the starting value within some short distance (generated usually by lines).
- Roof edge: a ridge edge where the intensity change is not instantaneous but occur over a finite distance (generated usually by the intersection of surfaces).

There are four general steps of edge detection:

(1) Smoothing: suppress as much noise as possible, without destroying the true edges.
(2) Enhancement: apply a filter to enhance the quality of the edges in the image (sharpening).
(3) Detection: determine which edge pixels should be discarded as noise and which should be retained (usually, thresholding provides the criterion used for detection).
(4) Localization: determine the exact location of an edge (sub-pixel resolution might be required for some applications, that is, estimate the location of an edge to better than the spacing between pixels). Edge thinning and linking are usually required in this step.

Edge detection using *derivatives*. As an image is a 2D function, operators describing edges are expressed using *partial derivatives*. Points which lie on an edge can be detected by: (1) detecting local maxima or minima of the first derivative or (2) detecting the zero-crossing of the second derivative. To compute the derivative of a sginl, we can approximate the derivative by finite differences:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x) \quad (h = 1) \tag{1}$$

This corresponds with a mask of $[-1, 1]$. The second derivative can be computed as:

$$\begin{aligned} f''(x) &= \lim_{h\to 0} \frac{f'(x+h) - f'(x)}{h} \approx f'(x+1) - f'(x) = f(x+2) - 2f(x+1) + f(x) \quad (h=1) \\ f''(x) &\approx f(x+1) - 2f(x) + f(x-1) \end{aligned} \tag{2}$$

Above, we simply changed the centering from $x+1$ to x. The mask for this operation is: $[1, -2, 1]$. Edge detection can also be performed using the gradient, which is a vector with magnitude and direction, defined as follows:

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{3}$$

The magnitude and direction of this vector are given by:

$$magn(\nabla f) = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} = \sqrt{M_x^2 + M_y^2} \tag{4}$$

$$dir(\nabla f) = \tan^{-1}\left(\frac{M_y}{M_x}\right) \tag{5}$$

We generally simplify the magnitude as $magn(\nabla f) \approx |M_x| + |M_y|$. The magnitude of gradient provides information about the strength of the edge. The direction of gradient is always perpendicular to the direction of the edge (the edge direction is rotated with respect to the gradient direction by -90 degrees). We can estimate the gradient with finite differences:

$$\begin{aligned} \frac{\partial f}{\partial x} &= \lim_{h_x\to 0} \frac{f(x+h_x, y) - f(x,y)}{h_x} \approx f(x+1, y) - f(x,y), \quad (h_x = 1) \\ \frac{\partial f}{\partial y} &= \lim_{h_y\to 0} \frac{f(x, y+h_y) - f(x,y)}{h_y} \approx f(x, y+1) - f(x,y), \quad (h_y = 1) \end{aligned} \tag{6}$$

Using pixel-coordinate notation, where j corresponds to the x direction and i to the negative y direction):

$$\begin{aligned} \frac{\partial f}{\partial x} &= f(i, j+1) - f(i,j) \\ \frac{\partial f}{\partial y} &= f(i-1, j) - f(i,j) = f(i,j) - f(i+1,j) \end{aligned} \tag{7}$$

The Roberts edge detector builds from this approximation and defines two masks M_x and M_y to implement this approximation:

$$M_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad M_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{8}$$

In this case, M_x and M_y are approximations at $(i+1/2, j+1/2)$. Other edge detectors have been identified as well. Consider the arrangement of pixels about the pixel (i,j):

$$\begin{matrix} a_0 & a_1 & a_2 \\ a_7 & [i,j] & a_3 \\ a_6 & a_5 & a_4 \end{matrix}$$

The partial derivatives can be computed by:

$$\begin{aligned} M_x &= (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \\ M_y &= (a_6 + ca_5 + a_4) - (a_0 + ca_1 + a_2) \end{aligned} \tag{9}$$

This is known as the **Prewitt** edge detector. The constant c implies the emphasis given to pixels closer to the center of the mask. In the case of $c = 1$:

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{10}$$

In the case of $c = 2$, we obtain the **Sobel** operator.

$$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad M_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{11}$$

The main steps in an edge detection using masks:

(1) Smooth the input $\hat{f}(x, y) = (f(x, y) * G(x, y))$
(2) $\hat{f}_x(x, y) = \hat{f}(x, y) * M_x(x, y)$
(3) $\hat{f}_y(x, y) = \hat{f}(x, y) * M_y(x, y)$
(4) $magn(x, y) = |\hat{f}_x| + |\hat{f}_y|$
(5) $dir(x, y) = \tan^{-1}(\hat{f}_y/\hat{f}_x)$
(6) If $magn(x, y) > T$, then possible edge point.

The magnitude of a gradient is an isotropic operator, detecting edges in any direction. Practical issues include: (1) The differential masks act as high-pass filters which tend to amplify noise. (2) To reduce the effects of noise, the image needs to be smoothed first with a low- pass filter. (3) The noise suppression-localization tradeoff: a larger filter reduces noise, but worsens localization (i.e., it adds uncertainty to the location of the edge) and vice versa. (4) Thresholding. (5) Edge thinning and linking are required to obtain good contours.

The criteria for an optimal edge detection include:

(1) Good Detection: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges).
(2) Good localization: the edges detected must be as close as possible to the true edges.
(3) Single Response Constraint: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge (created by noise).

Edge detection can be detected by finding the zero-crossings of the second derivative. In general, there are two operators in 2D that correspond to the second derivative: (1) Laplacian, (2) Second directional derivative. Consider the Laplacian:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{12}$$

We can approximate $\nabla^2 f$ as follows:

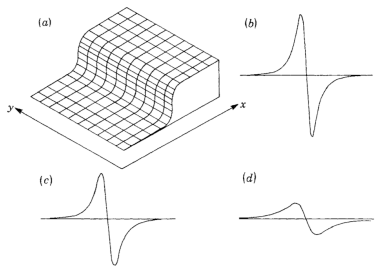
$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} &= f(i, j+1) - 2f(i, j) + f(i, j-1) \\ \frac{\partial^2 f}{\partial y^2} &= f(i+1, j) - 2f(i, j) + f(i-1, j) \\ \nabla^2 f &= -4f(i, j) + f(i, j+1) + f(i, j-1) + f(i+1, j) + f(i-1, j) \end{aligned} \tag{13}$$

The Laplacian is an isotropic operator, cheaper to implement (one mask only), but does does not provide information about edge direction and is more sensitive to noise. The reduce the noise effect, the image is first smoothed with a low-pass filter. In the case of the LOG (Laplacian of Gaussian), the low pass filter is chosen to be a Gaussian:

$$G(x, y) = \exp\Big(-\frac{x^2 + y^2}{2\sigma^2}\Big) \tag{14}$$

It can be shown that

$$\begin{aligned}\nabla^2[f(x,y) * G(x,y)] &= \nabla^2 G(x,y) * f(x,y) \\ \nabla^2 G(x,y) &= \Big(\frac{r^2-\sigma^2}{\sigma^4} e^{-r^2/2\sigma^2}\Big), \quad (r^2 = x^2+y^2)\end{aligned} \tag{15}$$



Marr-Hildreth Operator

The Marr-Hildreth operator involves determination of zero crossings of the second directional derivative. It is valid at multiple scale and reasonable spacial localization. It can be shown that the most optimal smoother that will satisfy both spatial and frequency localization is the Gaussian. Recall,

$$\begin{aligned}G(x) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(-\frac{x^2}{2\sigma^2}\Big) \\ \text{FT: } \overline{G}(\omega) &= \exp\Big(-\frac{1}{2}\sigma^2\omega^2\Big)\end{aligned} \tag{16}$$

We must have $\Delta x \Delta\omega >$ SE, where SE is some standard error. In 2D,

$$G(\overrightarrow{r}) = \frac{1}{\pi\sigma^2} \exp\Big(-\frac{r^2}{2\sigma^2}\Big), \quad r = \sqrt{x^2+y^2} \tag{17}$$

From Logan's theorem, it is well known that that zero crossings of the second derivative have important information. Let,

$$f(x,y) = D^2[G(\overrightarrow{r}) * I(x,y)] = D^2 G * I(x,y) \tag{18}$$

D^2G is known as the LOG (Laplacian of Gaussian). In 1D,

$$G''(x) = -\frac{1}{\sigma(2\pi)^{1/2}}\Big(1-\frac{x^2}{\sigma^2}\Big)\exp\Big(-\frac{x^2}{2\sigma^2}\Big) \tag{19}$$

Replacing D^2 with ∇^2 (Laplacian operator), we obtain the Marr-Hildreth LOG operator:

$$\nabla^2 = -\frac{1}{\pi\sigma^4}\Big(1-\frac{r^2}{2\sigma^2}\Big)\exp\Big(-\frac{r^2}{2\sigma^2}\Big) \tag{20}$$

Shown on the left are the values of the the first and second derivative of the LOG for an idealized step function.

Calculus of Variations

In calculus of variations, we look for extrema of expressions that depend on functions rather than parameters (i.e. *functionals*). Consider the integral

$$I = \int_{x_1}^{x_2} F(x, f, f')dx \tag{21}$$

Here F depends on the unknown function f and its derivative f'. Let us assume that $f(x_1) = f_1$ and $f(x_2) = f_2$. Suppose that the function $f(x)$ is a solution of the extremum problem. Then we expect that small variations in $f(x)$ should not change the integral significantly. Let $\eta(x)$ be some test function. If we add $\epsilon\eta(x)$ to $f(x)$, we expect that the integral will change by an amount proportional to ϵ^2 for small values of ϵ. If, instead, it varied linearly with ϵ, we could increase or decrease the integral as desired and would therefore not be at extrememum. We want

$$\left.\frac{dI}{d\epsilon}\right|_{\epsilon=0} = 0 \tag{22}$$

This we must be true for all test functions $\eta(x)$.

In our problem, we must have $\eta(x_1) = \eta(x_2) = 0$ to satisfy the boundary conditions. Substituting the addition of the test function, where $\eta'(x) = d\eta/dx$,

$$I = \int_{x_1}^{x^2} F(x, f + \epsilon\eta, f' + \epsilon\eta')dx \tag{23}$$

If F is suitably differentiable, we can expand the integrand in a Taylor series,

$$F(x, f + \epsilon\eta, f' + \epsilon\eta') = F(x, f, f') + \epsilon\frac{\partial}{\partial f}F(x, f, f')\eta(x) + \epsilon\frac{\partial}{\partial f'}F(x, f, f')\eta'(x) + e \tag{24}$$

where e consists of terms in higher powers of ϵ. Thus,

$$I = \int_{x_1}^{x_2} (F + \epsilon\eta(x)F_f + \epsilon\eta'(x)F_{f'} + e)dx \tag{25}$$

Differentiating with respect to ϵ and setting ϵ equal to zero yields

$$0 = \int_{x_1}^{x_2} (\eta(x)F_f + \eta'(x)F_{f'})dx \tag{26}$$

We can simply this integral using integration by parts:

$$\int_{x_1}^{x_2} \eta'(x)F_{f'}dx = [\eta(x)F_{f'}]_{x_1}^{x_2} - \int_{x_1}^{x_2} \eta(x)\frac{d}{dx}F_{f'}dx \tag{27}$$

From our specified boundary conditions $\eta(x_1) = \eta(x_2) = 0$, $[\eta'(x)F_{f'}]_{x_1}^{x_2} = 0$. Substituting our above expression, we obtain,

$$0 = \int_{x_1}^{x_2} \eta(x)\Big(F_f - \frac{d}{dx}F_{f'}\Big)dx \tag{28}$$

As this must be true for all test functions $\eta(x)$, then

$$\boxed{F_f - \frac{d}{dx}F_{f'} = 0} \tag{29}$$

This is called the *Euler equation* for this problem. We can generalize this method in a number of ways. For instance, if we are not provided the boundary conditions $f(x_1) = f_1$ and $f(x_2) = f_2$, then in other for the term $[\eta(x)F_{f'}]_{x_1}^{x_2}$ to be 0 for all $\eta(x)$ we must introduce the so-called *natural* boundary conditions:

$$F_{f'} = 0 \quad \text{at } x = x_1 \text{ and } x = x_2 \tag{30}$$

Next, for the case when the integrand contains higher derivatives:

$$I = \int_{x_1}^{x_2} F(x, f, f', f'', ...), \tag{31}$$

the Euler equation generalizes to:

$$F_f - \frac{d}{dx}F_{f'} + \frac{d^2}{dx^2}F_{f''} = 0 \tag{32}$$

In this case, we must specify the boundary values of all but the higher derivatives. We can also consider the case when the integrand depends on several functions $f_1(x), f_2(x), ...$:

$$I = \int_{x_1}^{x_2} F(x, f_1, f_2, ..., f_1', f_2', ...)dx \tag{33}$$

In this case, the Euler equations are:

$$F_{f_i} - \frac{d}{dx} F_{f_i'} = 0 \tag{34}$$

Next, consider the case when there are two independent variables x and y, and we are able to find a function $f(x, y)$ that yields an extremum of the integral

$$I = \iint_D F(x, y, f, f_x, f_y) dx dy \tag{35}$$

Here f_x and f_y are the partial derivatives of f with respect to x and y, respectively, and the integral is over some simply-connected closed region D. We introduce a test function $\eta(x, y)$ and add $\epsilon\eta(x, y)$ to $f(x, y)$. We are given the values of $f(x, y)$ on the boundary ∂D of the region, so the test function must be zero on the boundary. Taylor expansion yields,

$$\begin{aligned} F(x, y, f + \epsilon\eta, f_x + \epsilon\eta_x, f_y + \epsilon\eta_y) = F(x, y, f, f_x, f_y) + \epsilon \frac{\partial}{\partial f} F(x, y, f, f_x, f_y)\eta(x, y) \\ + \epsilon \frac{\partial}{\partial f_x} F(x, y, f, f_x, f_y)\eta_x(x, y) + \epsilon \frac{\partial}{\partial f_y} F(x, y, f, f_x, f_y)\eta_y(x, y) + e \end{aligned} \tag{36}$$

where e consists of terms in higher powers of ϵ. Thus,

$$I = \iint_D (F + \epsilon\eta F_f + \epsilon\eta_x F_{f_x} + \epsilon\eta_y F_{f_y}) dx dy \tag{37}$$

and differentiating with respect to ϵ and setting ϵ equal to zero yields

$$0 = \iint_D (\eta F_f + \eta_x F_{f_x} + \eta_y F_{f_y}) dx dy \tag{38}$$

By Gauss's theorem,

$$\iint_D \Big(\frac{\partial Q}{\partial x} + \frac{\partial P}{\partial y}\Big) dx dy = \int_{\partial D} (Q dy - P dx), \tag{39}$$

so that,

$$\iint_D \Big(\frac{\partial}{\partial x}(\eta F_{f_x}) + \frac{\partial}{\partial y}(\eta F_{f_y})\Big) = \int_{\partial D} (\eta F_{f_x} dy - \eta F_{f_y} dx) \tag{40}$$

Given the boundary conditions, the term on right must be zero, so that

$$\iint_D (\eta_x F_{f_x} + \eta_y F_{f_y}) dx dy = -\iint_D \Big(\eta \frac{\partial}{\partial x} F_{f_x} + \eta \frac{\partial}{\partial y} F_{f_y}\Big) dx dy \tag{41}$$

Consequently,

$$0 = \iint_D \eta\Big(F_f - \frac{\partial}{\partial x} F_{f_x} - \frac{\partial}{\partial y} F_{f_y}\Big) dx dy \tag{42}$$

for all test functions η. We must have, then, that

$$F_f - \frac{\partial}{\partial x} F_{f_x} - \frac{\partial}{\partial y} F_{f_y} = 0 \tag{43}$$

Here, the Euler equation is a partial differential equation. An immediate extension is to the case in which the value of f on the boundary ∂D is not specified. For the integral,

$$\int_{\partial D} \eta(F_{f_x} dy - F_{f_y} dx) \tag{44}$$

to be zero for all test functions η, we must have

$$F_{f_x}\frac{dy}{ds} = F_{f_y}\frac{\partial x}{\partial s} \tag{45}$$

where s is a parameter that varies along the boundary. The extension to more than two independent variables is also immediate.

Variational Problems with Constraints

A problem in the calculus of variations can also have constraints. Suppose, for example, that we want to find an extremum of

$$I = \int_{x_1}^{x_2} F(x, f_1, f_2, ..., f_n, f_1', f_2', ..., f_n')dx \tag{46}$$

subject to the constraints $g_i(x, f_1, f_2, ..., f_n) = 0$ for $i = 1, 2, ..., m$ with $m < n$. We can solve the modified Euler equations

$$\frac{d\Phi}{df_i} - \frac{d}{dx}\frac{d\Phi}{df_i'} = 0 \quad \text{for } i = 1, 2, ..., n \tag{47}$$

subject to the constraints. Here

$$\Phi \equiv F + \sum_{i=1}^{m} \lambda_i(x) g_i(x, f_1, f_2, .., f_n) \tag{48}$$

The unknown functions $\lambda_i(x)$ are called *Lagrange multipliers*. Constrains in the form of integrals are treated similarly. Supposed we want to find an extremum of the integral

$$I = \int_{x_1}^{x_2} F(x, f_1, f_2, ..., f_n, f_1', f_2', ..., f_n')dx \tag{49}$$

subject to the constraints

$$\int_{x_1}^{x_2} g(x, f_1, f_2, ..., f_n, f_1', f_2', ..., f_n')dx = c_i \quad \text{for } i = 1, 2, ..., m, \tag{50}$$

where c_i are given constants. We now solve the modified Euler equations:

$$\frac{d\Psi}{df_i} - \frac{d}{dx}\frac{d\Psi}{df_i'} = 0 \quad \text{for } i = 1, 2, ..., m \tag{51}$$

subject to the constraints. Here

$$\Psi = F + \sum_{i=1}^{m} \lambda_i g_i(x, f_1, f_2, ..., f_n) \tag{52}$$

Canny Edge Detector

The objective of any edge detection algorithm is to: (1) maximize detection accuracy, (2) improve localization performance, and (3) prevent multiple responses. Consider the task of step edge detection with Gaussian noise. Let,

$$I(x) = Au_{-1}(s) + n(x) \tag{53}$$

where

$$u_{-1}(x) = \begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases} \tag{54}$$

Let the impulse response of the edge operator we desire be $f(x)$. The output $O(x_0)$ for the input input $I(x)$ is given by the convolution below:

$$O(x_0) = \int_{-\infty}^{+\infty} I(x) f(x_0 - x) dx \tag{55}$$

We can use linearity of the convolution to split the output due to the signal and noise components of the input (i.e. $I(x) = Au_{-1}(x) + n(x)$). Assuming that the determined edge is centered around x_0, then the output due to the signal is:

$$\int_{-\infty}^{\infty} Au_{-1}(x) dx = A \int_{-\infty}^{0} f(x) dx \tag{56}$$

where $f(x)$ is the impulse response. As noise has mean zero, we determine the expected value of the noise component as:

$$E\Big[\int_{-\infty}^{\infty} f(x) n(-x) dx \Big]^2 \tag{57}$$

With this assumption of white noise (mean 0 and variance n_0, the above expression simplifies to:

$$E\Big[\int_{-\infty}^{\infty} f(x) n(-x) dx \Big]^2 = E\Big[\int_{-\infty}^{\infty} f^2(x) n^2(-x) dx \Big] = n_0^2 \int_{-\infty}^{\infty} f^2(x) dx \tag{58}$$

where $n_0^2 = E[n^2(x)]$ or all x. The signal to noise ratio (SNR) is therefore quotient of the signal and square root of the variance component above:

$$\text{SNR} = \frac{A \int_{-\infty}^{0} f(x) dx}{n_0 \sqrt{\int_{-\infty}^{\infty} f^2(x) dx}} = \frac{A}{n_0} \Sigma \quad \text{where } \Sigma = \frac{\int_{-\infty}^{0} f(x) dx}{\sqrt{\int_{-\infty}^{\infty} f^2(x) dx}} \tag{59}$$

The SNR above is an indicator of detection performance. We therefore would like to find the transfer function $f(x)$ to maximize SNR.

Localization: Localization in this case is the reciprocal of the standard deviation of the distance between the actual maximum and detected edge, given by the maxima of our output $O(x_0)$. In this case, the true edge is at $x = 0$, while the determined edge is at $x = x_0$ (i.e. zero crossing). x_0 therefore represents the distance between the true and calculated edge.

$$O'(x_0) = \frac{d}{dx_0} \int_{-\infty}^{\infty} f(x) I(x_0 - x) dx = \int_{-\infty}^{\infty} f'(x) I(x_0 - x) dx \tag{60}$$

As above, we can split $O'(x_0)$ into signal and noise components, denoted as O'_s and O'_n, respectively.

$$O'_s(x_0) = \int_{-\infty}^{\infty} f'(x) A u_{-1}(x_0 - x) dx = \int_{-\infty}^{x_0} A f'(x) dx = A f(x_0) \tag{61}$$

Similarly, the response of the noise component is:

$$E[O'^2_n(x)] = n_0^2 \int_{-\infty}^{\infty} f'^2(x) dx \tag{62}$$

Any function f can be split into a sum of symmetric and antisymmetric components. The symmetric component of f has no affect on the detection of localizing ability of the operator but will only contribute to the noise. We therefore add the constraint that f be antisymmetric to maximize SNR and localization. A Taylor expansion of $O'_s(x_0)$ about the origin yields:

$$O'_x(x_0) = A f(x_0) = A \sum_n \frac{f^n(0)}{n!} x^n \approx x_0 A f'(0) \tag{63}$$

as $f(0) = 0$. For a zero crossing of the derivative of the output (i.e. location of the maximum/edge),

$$O'(x_0) = O'_s(x_0) + O'_n(x_0) = 0 \tag{64}$$

which implies, $O'_s(x_0) = -O'_n(x_0)$ and $E[O'^2_s(x_0)] = E[O'^2_n(x_0)]$. As,

$$E[O'^2_s(x_0)] = E[x_0^2 A^2 f'(0)] = A^2 f'(0)^2 E[x_0^2] \tag{65}$$

we can develop a relationship between the expected value of the edge localized to x_0 and the operator transfer function:

$$\begin{aligned} A^2 f'(0)^2 E[x_0^2] &= n_0^2 \int_{-\infty}^{\infty} f'^2(x)dx \\ E[x_0^2] &\approx \frac{n_0^2 \int_{-\infty}^{\infty} f'^2(x)dx}{A^2 f'(0)^2} = \delta x_0^2 \end{aligned} \tag{66}$$

δx_0 is an approximation of the standard deviation of the distance of the actual maximum (x_0) to the true edge at $x = 0$. We define localization as the reciprocal of δx_0:

$$\text{Localization} = \frac{1}{\delta x_0} = \frac{A}{n_0} \frac{|f'(0)|}{\sqrt{\int_{-\infty}^{\infty} f'^2(x)dx}} = \frac{A}{n_0}\Lambda \quad \text{where } \Lambda = \frac{|f'(0)|}{\sqrt{\int_{-\infty}^{\infty} f'^2(x)dx}} \tag{67}$$

As we desire to maximize our localization and SNR criteria, we define a new parameters, which is the product:

$$\Sigma(f)\Lambda(f) = \frac{\int_{-\infty}^{0} f(x)dx}{\sqrt{\int_{-\infty}^{\infty} f^2(x)dx}} \frac{|f'(0)|}{\sqrt{\int_{-\infty}^{\infty} f'^2(x)dx}} \tag{68}$$

We next want to show that this criterion is scale invariant. Consider the response $f_w = f_w(x) = f(x/w)$:

$$\Sigma(f_w)\Lambda(f_w)\left[\sqrt{w}\frac{\int_{-\infty}^{0} f(x)dx}{\sqrt{\int_{-\infty}^{\infty} f^2(x)dx}}\right]\left[\frac{1}{\sqrt{w}} \frac{|f'(0)|}{\sqrt{\int_{-\infty}^{\infty} f'^2(x)dx}}\right] \tag{69}$$

As shown above, SNR varies with $\sqrt{w}$, while localization varies inversely with $\sqrt{w}$. However, the product does not vary with scale, indicating that there is an operator with optimal performance regardless of spatial scaling.

To solve the optimization problem, we make the following simplifications: (1) We set all but one of the integrals to an undetermined constant and solve using Lagrange multiplies. We then find the extreme value of the remaining integral. (2) As f is antisymmetric, then we only need to consider the limits from $-\infty \to 0$ to simplify all other cases. Furthermore, this allows $f'(0)$ to be set as a boundary condition. We next set the lower limit of the integral to a finite $-W$ as we are dealing with an operator of finite extent. As a result, the problem reduces to finding the minimum of the integral in the denominator of the SNR term. We therefore wish to minimize the following integral:

$$\int_{-W}^{0} f^2(x)dx \tag{70}$$

subject to the constraints:

$$\begin{aligned} \int_{-W}^{0} f(x)dx &= c_1 \\ \int_{-W}^{0} f'^2(x)dx &= c_2 \\ \int f'(0) &= c_3 \end{aligned} \tag{71}$$

Assuming that we have the boundary conditions $f(0) = f(-W) = 0$ and that the continuous and differentiable up to the (n-1)th derivative over $(-\infty, \infty)$, then we can minimize the functional $F(x, f, f') = f^2(x)$ subject to the constraints $\int_a^b G_i(x, f, f') = c_i$. We can use Euler's equation here with the composite functional:

$$\Psi(x, f, f') = F(x, f, f') + \lambda_1 G_1(x, f, f') + \lambda_2 G_2(x, f, f') + ... = f^2 + \lambda_1 f'^2 + \lambda_2 f \tag{72}$$

The Euler equation for this problem is:

$$\Psi_f - \frac{d}{dx}\Psi_{f'} = 0 \tag{73}$$

Substituting our composite functional:

$$2f(x) - 2\lambda_1 f''(x) + \lambda_2 = 0 \tag{74}$$

The general form of the solution is the sum of the non-homogeneous and homogeneous solutions:

$$f(x) = -\frac{\lambda_2}{2} + a_1 e^{\alpha x} + a_2 e^{-\alpha x} \tag{75}$$

where $\alpha = 1/\sqrt{\lambda_1}$ and the constants are determined from the boundary conditions $f(0) = f(-W) = 0$. The general solution takes the form:

$$f(x) = -\frac{\lambda_2}{2}\left(1 - \frac{\cosh\alpha\left(x + \frac{W}{2}\right)}{\cosh\alpha\frac{W}{2}}\right) \tag{76}$$

The SNR and localization as a function of the parameters λ_1 and λ_2 for a width W of 2 yields:

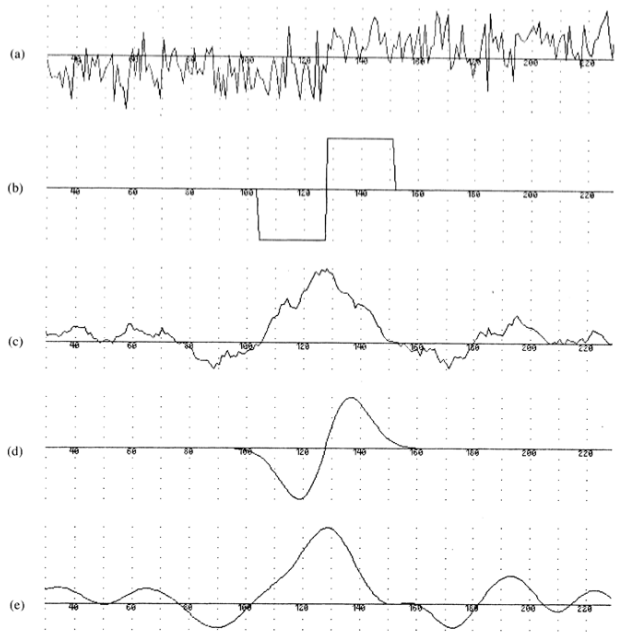
$$\Sigma = \frac{2\alpha\cosh\alpha - 2\sinh\alpha}{\sqrt{2\alpha^2\cosh 2\alpha - 3\alpha\sinh 2\alpha + 2\alpha^2}} \tag{77}$$

$$\Lambda = \frac{\alpha\sinh\alpha}{\sqrt{\alpha\sinh 2\alpha - 2\alpha^2}} \tag{78}$$

As α tends to 0, we see that $f(x) = -\lambda_2\alpha^2(1 - x^2)$, with subsequent SNR and localization:

$$\Sigma = \sqrt{\frac{5}{6}} \qquad \Lambda = \sqrt{\frac{3}{4}} \tag{79}$$

For the case $\alpha \to \infty$, we see that the SNR approaches 1, while the localization increases without bound. This indicates that the difference of boxes is the best possible filter over the range $[-W, W]$, which is in fact the optimal Wiener filter for the step edge. Below is an example of localization of the step edge with the Canny derived difference of box operators. We see that there are multiple peaks in the function and no single maximum. We therefore wish to develop a methodology to eliminate these excess maxima.



In order to prevent multiple responses, we add another criteria. Recall, from signal processing, Rice's Theorem, which states that the average distance between zero crossings of the response of a function g to Gaussian noise is:

$$x_{avg} = \pi\Big(\frac{-R(0)}{R''(0)}\Big)^{1/2} \quad \text{where } R(\tau) = \int_{-\infty}^{\infty} g(x)g(x-\tau)dx \tag{80}$$

Note: $R(\tau)$ is the autocorrelation function. As,

$$R(0) = \int_{-\infty}^{\infty} g^2(x)dx \quad \text{and} \quad R''(0) = -\int_{-\infty}^{\infty} g'^2(x)dx \tag{81}$$

then for our function $g = f'$, which we wish to optimize,

$$x_{zc}(f) = \pi\left(\frac{\int_{-\infty}^{\infty} f'^2(x)dx}{\int_{-\infty}^{\infty} f''^2(x)dx}\right) \tag{82}$$

The distance between adjacent maxima in the noise response of f is double the zero-crossing distance (i.e. $x_{\max}(f) = 2x_{zc}(f)$) must be some fraction k of the window width W, else the edge is not properly localized. Then $x_{\max}(f) = 2x_{zc}(f) = kW$. With this additional constraint, the new functional takes the form:

$$\Psi(x, f, f', f'') = f^2 + \lambda_1 f'^2 + \lambda_2 f''^2 + \lambda_3 f \tag{83}$$

The Euler equation generalizes to:

$$\sum_{n=0} \frac{d^n}{dx^n}\Psi_{f^n}(-1)^n = \Psi_f - \frac{d}{dx}\Psi_{f'} + \frac{d^2}{dx^2}\Psi_{f''} = 0 \tag{84}$$

Substituting our functional yields the following differential equation:

$$2f(x) - 2\lambda_1 f''(x) + 2\lambda_2 f''''(x) + \lambda_3 = 0 \tag{85}$$

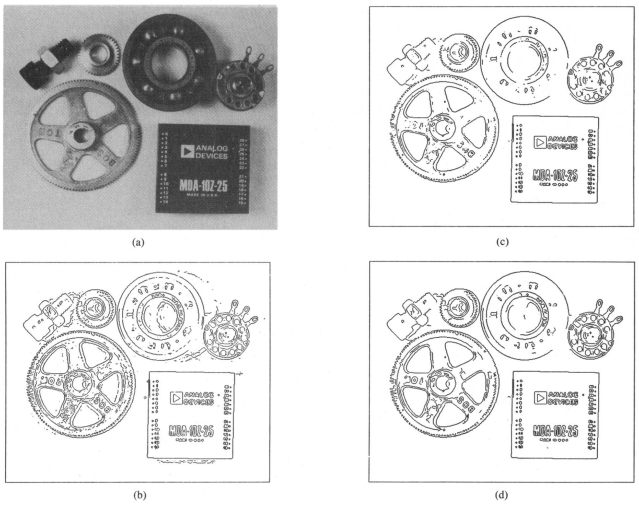
The solution to this differential equation is the sum of the non-homogeneous (constant) and homogeneous equations (sum of exponentials for the form $e^{\gamma x}$). Then

$$
\begin{aligned}
2 - \lambda_1\gamma^2 + 2\lambda_2\gamma^4 &= 0 \\
\gamma^2 &= \frac{\lambda_1}{2\lambda_2} \pm \frac{\sqrt{\lambda_1 - 4\lambda_2}}{2\lambda_2}
\end{aligned}
\tag{86}
$$

This solution to this differential equation can be approximated with the **first derivative of the Gaussian**.

$$
f(x) = -\frac{x}{\chi^2} \exp\Big(-\frac{x^2}{2\sigma^2}\Big) \tag{87}
$$

Canny further suggested two thresholds, T_1 and T_2 to account for hysteresis (shown on the bottom left of the figure on the next page):



Noise Estimation: We use the Wiener filtering method to optimally estimating one component of the signal. Let $g_1(x)$ be the edge and $g_2(x)$ be the noise; paradoxically, $g_2(x)$ is what we desire. Recall that we can define the correlation as:

$$
R_{ij}(\tau) = \int_{-\infty}^{\infty} g_i(x) g_j(x+\tau) dx \tag{88}
$$

We assume that the signal and noise are uncorrelated, so $R_{12}(\tau) = 0$. The optimal Wiener filter is implicitly defined as:

$$
R_{11}(\tau) = \int_{-\infty}^{\infty} (R_{11}(\tau - x) + R_{22}(\tau - x)) K(x) dx. \tag{89}
$$

where $K(x)$ is the optimal filter. The autocorrelation of a filter in response to white noise is equal to the autocorrelation of its impulse response (i.e. Transfer Function), then assuming a first derivative of a Gaussian:

$$
\begin{aligned}
R_{11}(x) &= k_3\Big(\frac{x^2}{2\sigma^2} - 1\Big) \exp\Big(-\frac{x^2}{4\sigma^2}\Big) \\
R_{22}(x) &= k_2 \exp\Big(-\frac{x^2}{4\sigma^2}\Big)
\end{aligned}
\tag{90}
$$

K can be approximated by a Gaussian given that the edge is large compared to the noise (as $R_1 1$ is the second derivative of the Gaussian).

Two or More Dimensions: In the case of multiple dimensions, the optimal detector is:

$$G = \exp\Big(- \frac{x^2 + y^2}{2\sigma^2} \Big) \tag{91}$$

and

$$G_n = \frac{\partial G}{\partial n} = n \cdot \nabla G \tag{92}$$

n should be oriented normal to the direction of the edge detected. We can estimate this from the smoothed gradient direction:

$$n = \frac{\nabla (G * I)}{|\nabla (G * I)|} \tag{93}$$

At an edge (local maximum), we have:

$$\frac{\partial}{\partial n} G_n * I = \frac{\partial^2}{\partial n^2} G * I = 0 \tag{94}$$

At this point, the edge strength will be:

$$|G_n * I| = |\nabla (G * I)| \tag{95}$$

Non Maximum Suppression

Non maximum suppression is an edge thinning technique. Given estimates of the image gradients, a search is carried out to determine if the gradient magnitude assumes a local maximum in the gradient direction. In some implementations, the algorithm categorizes the continuous gradient directions into a small set of discrete directions, and then moves a 3x3 filter over the output of the previous step (that is, the edge strength and gradient directions). At every pixel, it suppresses the edge strength of the center pixel (by setting its value to 0) if its magnitude is not greater than the magnitude of the two neighbors in the gradient direction. For example,

- if the rounded gradient angle is zero degrees (i.e. the edge is in the north–south direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the east and west directions,
- if the rounded gradient angle is 90 degrees (i.e. the edge is in the east–west direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north and south directions,
- if the rounded gradient angle is 135 degrees (i.e. the edge is in the northeast–southwest direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north west and south east directions,
- if the rounded gradient angle is 45 degrees (i.e. the edge is in the north west–south east direction) the point will be considered to be on the edge if its gradient magnitude is greater than the magnitudes at pixels in the north east and south west directions.

In more accurate implementations, linear interpolation is used between the two neighbouring pixels that straddle the gradient direction. For example, if the gradient angle is between 45 degrees and 90 degrees, interpolation between gradients at the north and north east pixels will give one interpolated value, and interpolation between the south and south west pixels will give the other (using the conventions of last paragraph). The gradient magnitude at the central pixel must be greater than both of these for it to be marked as an edge. Note that the sign of the direction is irrelevant, i.e. north–south is the same as south–north and so on.

Hough Transform

Edges can be linked to form lines. One such example is the Hough Transform. Briefly, Hough space is the transformation an image (x, y) to a space with coordinates (m, b), where m and b are the slope and intercepts of set of lines that describe the image. For $m \in (-\infty, \infty)$, then

$$\rho = -x \cos\theta + y \sin\theta \quad \theta \in [0, 2\pi], \rho \in [0, \rho_{\max}] \tag{96}$$

The basic Hough transform algorithm involves the following steps:

(1) Initialize $H[d, \theta] = 0$
(2) For each Edge point in $I[x, y]$, $D[d, \theta] = 1$
(3) Find values of (d, θ) where $H[d, \theta]$ is maximum
(4) The line detected in the image is given by $d = x \cos\theta + y \sin\theta$

We can generalize this procedure to define any shape, circle, ellipse, etc. Other line detecting algorithms include the Navatia-Babu line finder, the Horn-Binford line finder, Burns line finder, and others.

Harris Operator

We next consider a detector to find points of interest and corners. Many such operators for corner detection exist, including the Moravec and Harris operators, though we will consider the Harris operator here. Vision tasks such as stereo and motion estimation require finding corresponding features across two or more views. Corners are good features to match to patches as there are large variations in the neighborhood. Let

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{97}$$

where $w(x, y)$ is a window function (and can include Gaussian smoothing as well), $I(x + u, y + v)$ is the shifted intensity, and $I(x, y)$ is the image intensity. For nearly constant patches, $E(u, v) = 0$. For distinctive patches, $E(u, v)$ will be much larger. A first order Taylor expansion of the shifted intensity yields

$$f(x + u, y + v) = f(x, y) + uf_x(x, y) + vf_y(x, y) + ... \approx f(x, y) + uf_x(x, y) + vf_y(x, y) \tag{98}$$

In a first order approximation, the Harris corner detector is:

$$\begin{aligned} \sum [I(x + u, y + v) - I(x, y)]^2 &\approx \sum [I(x, y) + uI_x + vI_y - I(x, y)]^2 \\ &= \sum (u^2 I_x^2 + 2uvI_x I_y + v^2 I_y^2) \\ &= \sum \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned} \tag{99}$$

For small shifts in $[u, v]$ we have a bilinear approximation:

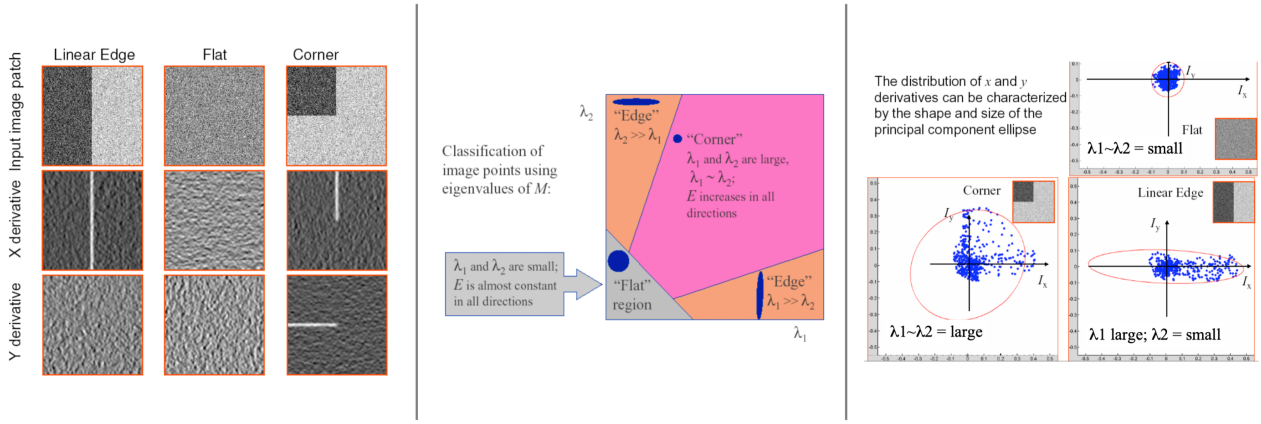
$$E(u, v) \cong \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{100}$$

where M is a 2x2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{101}$$

The eigenvalues of M hold significant meaning, as shown in the image below. Let λ_1 and λ_2 be the eigenvalues of M. We have the following cases:

- λ_1, λ_2 small: Flat Region
- λ_1, λ_2 large: Corner
- λ_1 large, λ_2 small: Edge
- λ_1 small, λ_2 large: Edge



Corner Measure Response: The corner measure response of the Harris operator is defined as:

$$R = \det M - k(\text{trace}M)^2 \tag{102}$$

where $\det M = \lambda_1\lambda_2$ and trace M $= \lambda_1 + \lambda_2$. k is an empirically determined constant, usually between 0.04 and 0.06.

Scale Invariant Feature Transform (SIFT)

The SIFT operator allows for multi-scale processing, including object segmentation, compression, texture analysis, etc. The scale space produces multiple resolutions. As we may look at pictures in various scales, etc., we desire an algorithm that is robust to invariant scales, illuminations, and rotations. This is primarily useful for matching images, patterns, etc. Invariance, here implies that the algorithm is invariant to certain values of scales, though it is used loosely.

We use Gaussian smoothing to transform an image into scale space. Consider a Gaussian filter with variance σ, denoted $G(x, y, \sigma)$ and an image $I(x, y)$. We define the convolution of the Gaussian with the image as $L(x, y, \sigma)$, where

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{103}$$

where,

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\Big(- \frac{1}{2\sigma^2}(x^2 + y^2)\Big) \tag{104}$$

To efficiently detect stable, key point location in scale space, we will look at the difference of the Gaussian function and its extrema. Let

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{105}$$

However, the above procedure is not efficient. Recall, $\sigma^2\nabla^2 G$ is a close approximation to a difference of Gaussians. The maxima and minima of this function produces the most stable image features compared to the gradient, Hessian and Harris operators. The relationship between the difference of Gaussians D and the Laplacian can be understood from the heat equation:

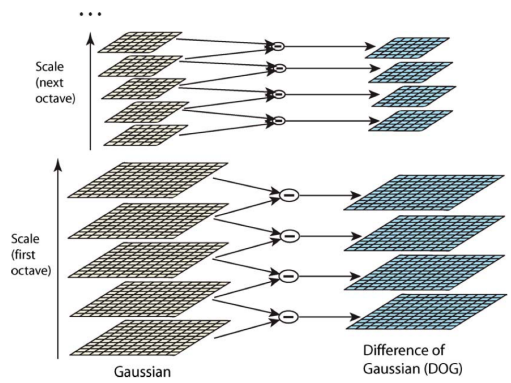
$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G \tag{106}$$

Substituting in,

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{107}$$

Therefore,

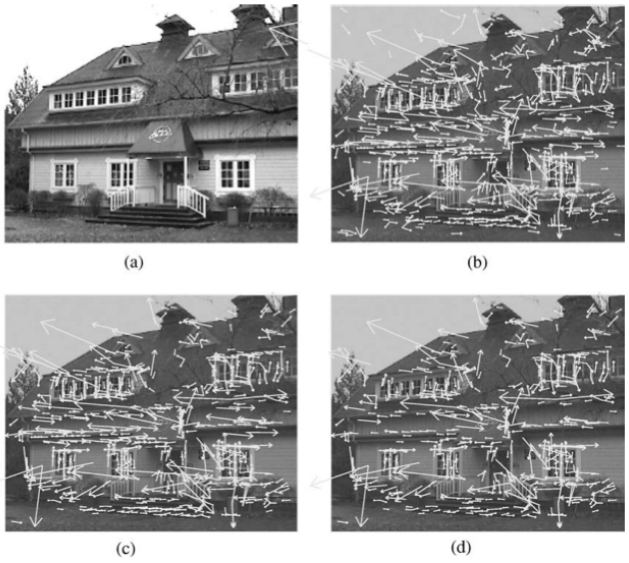
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \nabla^2 G \tag{108}$$



We define some σ and then fix some k, which is empirically determined to be $k = \sqrt{2}$. We then vary over a reasonable range of σ and k to fully determine the difference of Gaussian function. Commonly, for every octave (increase in σ by a factor of 2), we perform the operation for approximately 5 values of k. A summary of this procedure is shown on the left.

To find the local extrema, we take neighbors in 3x3 region surrounding the pixel of interest across the scale above and below, for a total of 26 neighbors. The pixel is only selected if it is the minima and the maxima.

Shown below is an example of the output of the SIFT process and identification of pixels of interest



We now wish to improve the accuracy of keypoint localization. We first take the difference of Gaussian that we calculated and expand it using a Taylor series expansion up to the second order:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{109}$$

where D and its derivatives are evaluated at the sample point and $\mathbf{x} = (x, y, \sigma)$. The location of the extremum, $\hat{\mathbf{x}}$ is determined by taking the derivative of $D(\mathbf{x})$ with respect to $\mathbf{x}$ and setting it to zero. This yields,

$$\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2}\frac{\partial D}{\partial \mathbf{x}} \tag{110}$$

Substituting $\hat{\mathbf{x}}$ into the original expression yields:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}} \tag{111}$$

This method allows for sub-pixel accuracy detection using a Taylor series. Furthermore, the above substitution allows for keypoint reduction.

In order to account for the edge response of the difference of Gaussian function, recall that a poorly defined peak in the function will have a large principal curvature across the edge but a small one in the perpendicular direction. We can construct the the Hessian matrix $\mathbf{H}$ at the location and scale of the keypoint as:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{112}$$

The eigenvalues of $\mathbf{H}$ are proportional other principal curvature of F. Let α and β be the eigenvalues with largest and smallest magnitude, respectively. The trace and determinant are:

$$\begin{aligned} \text{Tr}(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta \\ \det(\mathbf{H}) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \end{aligned} \tag{113}$$

Let r be the ratio of the two eigenvalues such that $\alpha = r\beta$, $r > 1$. Then,

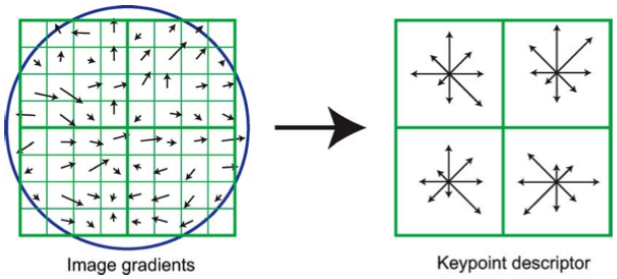
$$\frac{\text{Tr}(\mathbf{H})^2}{\det(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{114}$$

For any image I, we can examine local derivatives and their eigenvalues to determine a topographic primal sketch, to assess whether locations in the image are peaks, saddles, bridge, flat surfaces, etc.

To determine orientation, we can simply calculate the magnitude and direction of the Gaussian smoothed image L with the closest scale to ensure scale invariant.

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y) + L(x,y+1) - L(x,y-1))^2} \\ \theta(x,y) &= \tan^{-1}((L(x,y+1) - L(x,y-1))/L(x+1,y) - L(x-1,y)) \end{aligned} \tag{115}$$

We can then create an orientation histogram weighted by the gradient magnitude, window, and the 1.5 times the scale of the keypoint. Keypoint descriptors and orientation histograms are created over 2x2 sample regions, created from 8x8 sets of samples, as shown in the image below:



We next justify why SIFT is insensitive to scale, illumination and view point (restricted). Recall that the SIFT operator is defined as:

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \tag{116}$$

where

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\Big(-\frac{1}{2\sigma^2}(x^2+y^2)\Big) \tag{117}$$

and the relationship between the difference of Gaussians and the Laplacian, determined by the heat equation is:

$$G(x,y,k\sigma) - G(x,y,\sigma) \approx (k-1)\sigma^2\nabla^2 G \tag{118}$$

Based on the provided task, we define the following transformations. Let $\mathcal{T}$ be an arbitrary translation, $\mathcal{R}$ an arbitrary image rotation, $\mathcal{H}$ be an affine transformation, and G be the Gaussian defined above. We know that all operators above weakly commute from fundamental linear algebra (i.e. $\mathcal{H}\mathcal{T} = \mathcal{T}'\mathcal{H}$) for an arbitrary $\mathcal{T}'$.

Furthermore, it is known that the rotation matrix and Gaussian operator strongly commute. Let G_δ represent the SIFT keypoint extraction operator. For an image $I(x, y)$ that undergoes, rotation, translation, and an affine transformation, the new image that undergoes the SIFT transformation is $I' = \mathcal{HTR}I$.

We next transform our image into scale by computing the difference of Gaussian operator. Note that as $G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G$, then

$$DI' = (k-1)\sigma^2\nabla^2 G * I' = (k-1)\sigma^2\nabla^2 G * (\mathcal{HTR}I) \tag{119}$$

As the we have introduced operators that weakly commute, then the set of scale space Laplacian extrema defined above is covariant to translations and rotations. As a result, the normalization scheme established by Lowe in his original methodology will effectively account for the new SIFT descriptors generated for I', thus canceling these operators. Normalization of direction will account for any rotation changes. For the case of linear illumination, we have a linear operator that weakly commutes, then normalization of the SIFT operator will account for changes there as well. For the case of non-linear illumination, Lowe normalizes his gradients after thresholding his gradients such that no gradient is larger than 0.2. This is one limitation. We have therefore shown that SIFT is insensitive to scale and illumination.

We next need to justify how SIFT is relatively insensitive to viewpoint. Let I_1 and I_2 be two images at different view points. Let D_1 and D_2 correspond their scale space transformations. We will assume here that the images have only Guassian blur, with and were taken at different sample rates. Assume we use a SIFT operator with σ^2. Let β^2 and δ^2 correspond to the additional variance from the view points that correspond to I_1 and I_2, respectively. Furthermore, let λ and μ correspond to differences in scale. The scale space for both images are:

$$D_1(x, y, \sigma) = D(\lambda x, \lambda y, \lambda\sqrt{\sigma^2 + \beta^2})$$
$$D_2(x, y, \sigma) = D(\mu x, \lambda y, \mu\sqrt{\sigma^2 + \delta^2})$$

For any point (x_0, y_0, s_0) in scale space that is a keypoint (i.e. the extrema of the Laplacian), and if $s_0 > \max(\lambda\beta, \mu\delta)$, then for each image at scale σ_1 and σ_2,

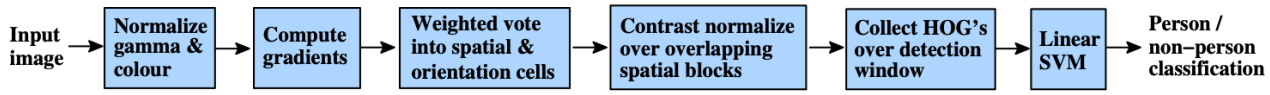
$$s_0 = \lambda\sqrt{\sigma_1^2 + \beta^2} = \mu\sqrt{\sigma_2^2 + \delta^2} \tag{120}$$

This shows that the SIFT descriptor around the sampling point (x, y, σ_1) is proportional to $\sqrt{\sigma_1^2 + c^2}$ for I_1 and $\sqrt{\sigma_2^2 + c^2}$ for I_2. Putting it altogether, consider two images at two different view points, $I_1' = D_{1,\beta}\mathcal{HTR}I_1$ and $I_2' = D_{2,\delta}\mathcal{HTR}I_2$. Assume that for these two images $\lambda < \mu$ (which are arbitrary view point distances), then we can say that the SIFT descriptors are identical. As above, we know that we can neglect the effects of rotation, translation, and illumination, based on their weak commutativity and the properties of the Laplacian. Furthermore, for an arbitrary key point (x_0, y_0, s_0), we know that there must exist a key point $(x_0/\lambda, y_0/\lambda, s_0)$ and $(x_0/\mu, y_0/\mu, s_0)$ for I_1 and I_2, respectively. The SIFT descriptors, as shown above are identical as $s_0 = \lambda\sqrt{\sigma_1^2 + \beta^2} = \mu\sqrt{\sigma_2^2 + \delta^2}$. As a result $c = \beta = \delta$ for this to hold. This can only be true if the blurs in each image are identical. We have thus shown that SIFT is insensitive to scale, illumination, and view point.

Intuitively, we know that the Laplacian is a scale invariant. Furthermore, as long as we detect stable features across all possible scales, as we do in SIFT, then we know that the image is scale invariant. Importantly, we made an assumption for a weakly commutative affine transformation. However, this is not always the case, and thus SIFT is not truly affine invariant.

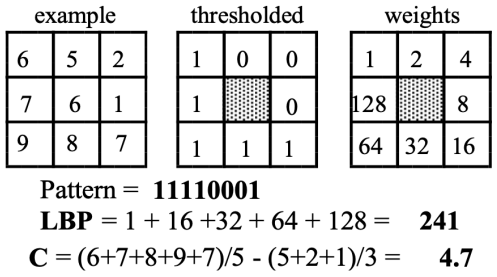
Histogram of Gradients (HOG)

HOG is considering the problem of pedestrian detection. HOG involves determination of simple gradients and calculation of subsequent histograms and training of an SVM for classification. Once the gradients are calculated, the number of gradient angles is calculated for an 8x8 cell, with weighting proportional to the magnitude of the gradient. A summary of the HOG pipeline is shown below:



Local Binary Patterns (LBPs)

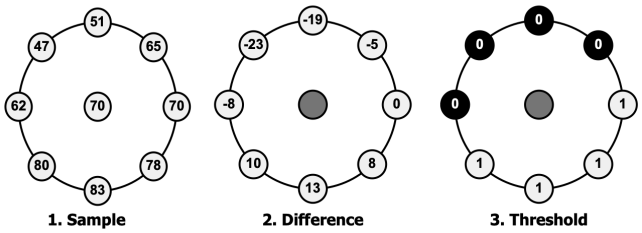
The objective of LBPs is for texture analysis. In general, the edge detectors discussed earlier as well as methods like SIFT and HOG would fail on textured images. Consider a 3x3 area of pixels. We then threshold relative to the center pixel and apply binary weights to the surrounding image, which is known as the local binary pattern, as shown to the right. We then average the pixels above and below the threshold. As a result, we have compressed the 3x3 region into two numbers, the LBP and C. LBP is invariant to any monotonic gray level change. The circular pattern allows for rotation invariance.



The value of an LBP code of a pixel (x_c, y_c) is given by:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \quad s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{121}$$

The radius of the region can be expanded for any general size, as shown on the here. In a texture pattern, the pixels are correlated. As a result of the correlation, one needs to be able to determine the covariance matrix of the pattern, which significantly increase complexity. Let the texture T be defined as the joint distribution of gray levels g_c and g_p, where $T = t(g_c, f_0, ..., g_{P-1})$. Subtracting g_c from g_p yields $T = t(g_c, g_0 - g_c, ..., g_{P-1} - g_C)$. Assuming that g_c is independent of $g_p - g_c$, then $T \sim t(g_c)t(g_0 - g_c, ..., g_{P-1} - g_c)$. $t(g_C)$ characterizes the overall luminance of the image, which is unrelated to texture, so $T \sim t(g_0 - g_c, ..., g_{P-1} - g_c)$, which is invariant. As the circular patterns are continuous, they are known as uniform detectors. Intuitively, these detectors represent spots, edge, corners, etc. depending on the distribution of the circular pattern. As a result, spatial rotation changes the LBP code. However, by going through all patterns, rotational invariance can be shown. Pattern identification is therefore just simple matching of the histograms. Fourier analysis can also be utilized to assess rotational shifts of LBPs.



The rotational invariance extension of the LBP is known as LBP-HF (Local Binary Pattern Histogram Fourier) features. Let (P, R) denote a circulate neighborhood, where P is the number of sampling points and R is the radius. The coordinates lie on the circle parameterized by (P, R) such that

$$(x_p, y_p) = \left(x + R\cos\left(\frac{2\pi p}{P}\right), y - R\sin\left(\frac{2\pi p}{P}\right)\right) \tag{122}$$

where $p \in \mathbb{N}$. The LBP label for a pixel at (x, y) is:

$$LBP_{P,R}(x, y) = \sum_{p=0}^{P-1} s(f(x, y) - f(x_p, y_p))2^p \tag{123}$$

where

$$s(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases} \tag{124}$$

We assume for the remainder of the problem that the LBP is uniform, meaning that it only contains two bitwise transitions from 0 to 1 or 1 to 0 and is circular. Let $U_P(n,r)$ be a uniform LBP pattern, with n being the number of 1's in the pattern and r the rotation. Let $I^{\theta\circ}(x,y)$ be the rotation of the image $I(x,y)$ by θ degrees. We denote the point (x',y') as the point on the original image (x,y), over which the LBP is centered. Note that $I^{\theta\circ}(x,'y') = I(x,y)$. In image space, we must have integer rotations. If there are P points in the LBP, then rotations must occur in integers of $360°/P$. Thus $\theta = k360°/P$, where $k = 0, 1, ..., P-1$.

We next consider how this quantization of rotation affects our uniform LBP pattern and our LBP histogram. Something rotated by $P+1$ steps will yield the same pattern as one rotated by 1 step, due to rotational symmetry. We can therefore replace r in our uniform pattern with $k + r \mod P$, as we only have P points and we are rotation above. Thus,

$$U_P(n,r) = U_P(n, r+k \mod P) \tag{125}$$

The histogram $h(U_P(n,r))$ is similarly shifted on its axis depending on its rotation. Let h represent the original histogram and h_θ be the histogram of the rotated image. Rotational symmetry here converts to a translation in the histogram. Therefore,

$$h_\theta(U_P(n, r+k)) = (U_P(n,r)) \tag{126}$$

where k is the integer shift. Clearly, a Fourier transform of our LBP histogram will result in a rotationally invariant pattern. Let $H(n,\cdot)$ be the discrete Fourier transform of our LBP histogram. Then

$$H(n,u) = \sum_{r=0}^{P-1} h(U_P(n,r)) e^{-i2\pi ur/P} \tag{127}$$

A rotational shift correlates with a phase shift in the DFT. Using our notation above, let $h'(U_P(n,r)) = h(U_P)(n, r-k)$. Then the DFT of h' is

$$H'(n,u) = H(n,u)e^{-i2\pi uk/P} \tag{128}$$

Let $\overline{H'(n,u)}$ is the complex conjugate of the DFT of our LBP operator. We desire some equivalency between H and H' as this demonstrates that the the DFT of the histogram is rotationally invariant. As $\overline{H'(n,u)} = \overline{H(n,u)e^{-i2\pi ur/P}} = \overline{H(n,u)}e^{i2\pi ur/P}$, then for any $1 \le n_1, n_2 \le P-1$

$$H'(n_1,u)\overline{H'(n_2,u)} = H(n_1,u)e^{-i2\pi ur/P}\overline{H(n_2,u)}e^{i2\pi ur/P} = H(n_1,u)\overline{H(n_2,u)} \tag{129}$$

We define,

$$\boxed{\text{LBP}^{u2}\text{-HF}(n_1,n_2,u) = H(n_1,u)\overline{H(n_2,u)}} \tag{130}$$

Based on our proof above, $\text{LBP}^{u2}\text{-HF}(n_1,n_2,u)$ is invariant to rotational invariance extension of the LBP.

Deep Learning

Neural networks have a long history which goes back to the first attempts to understand how the human and mammalian brain works and how what we call intelligence is formed. From a physiological point of view, one can trace the beginning of the field back to the work of Santiago Ramon y Cajal, who discovered that the basic building element of the brain is the neuron. The brain comprises approximately 60-100 billions neurons; that is, a number of the same order as the number of stars in our galaxy! Each neuron is connected with other neurons via elementary structural and functional units/links, known as synapses. It is estimated that there are 50-100 trillions of synapses. These links mediate information between connected neurons. The most common type of synapses are the chemical ones, which convert electric pulses, produced by a neuron, to a chemical signal and then back to an electrical one. Depending on the input pulse(s), a synapse is either activated or inhibited. Via these links, each neuron is connected to other neurons and this happens in a hierarchical way, in a layer-wise fashion.

A milestone from the learning theory's point of view occurred in 1943, when Warren McCulloch and Walter Pitts developed a computational model for the basic neuron. Moreover, they provided results that tie neurophysiology with mathematical logic. They showed that given a sufficient number of neurons and adjusting appropriately the synaptic links, each one represented by a weight, one can compute, in principle, any computable function. As a matter of fact, it is generally accepted that this is the paper that gave birth to the fields of neural networks and artificial intelligence. Pattern classifiers are generally considered non-linear mappings from patterns to classifiers. Frank Rosenblatt borrowed the idea of a neuron model, as suggested by McCulloch and Pitts, and proposed a true learning machine, which learns from a set of training data. Rosenblott separated two classes using linear hyperplanes. In the most basic version of operation, he used a single neuron and adopted a rule that can learn to separate data, which belong to two linearly separable classes. That is, he built a Pattern Recognition system. He called the basic neuron a perceptron and developed a rule/algorithm, the perceptron algorithm, for the respective training. The perceptron will be the kick-off point for our tour in this series of lectures.

One starting point is the simple problem of a linearly separable two-class (ω_1, ω_2) classification task. In other words, we are given a set of training samples $(y_n, x_n), n = 1, 2, ..., N$, with $y \in [-1, +1]$, and it is assumed that there exists a hyperplane,

$$\begin{aligned} &\theta_*^T x = 0 \quad \text{such that,} \\ &\theta_*^T x > 0, \quad \text{if } x \in \omega_1 \\ &\theta_*^T x < 0, \quad \text{if } x \in \omega_2 \end{aligned} \tag{131}$$

In other words, such a hyperplane classifies correctly all the points in the training set. For notational simplification, the bias term of the hyperplane has been absorbed in θ_*^T. However, we need some errors in the training set in order to improve generalizability. With just the bias and the variance, we can make non-linear operations of x_i, which we then compress into y_i. We are linear in y_i, which are non-linear functions of x_i.

The goal now becomes that of developing an algorithm that iteratively computes a hyperplane that classifies correctly all the patterns from both classes. To this end, a cost function must first be adopted. Let the available estimate at the current iteration step of the unknown parameters be θ. Then, there are two possibilities: (1) all points are classified correctly; this means that a solution has been obtained. (2) θ classifies correctly some of the points and the rest are misclassified. Let y be the set of all misclassified samples. The perceptron cost is defined as:

$$J(\theta) = -\sum_{n: s_n \in y} y_n \theta^T x_n \tag{132}$$

where

$$y_n = \begin{cases} +1 & \text{if } x \in \omega_1 \\ -1 & \text{if } x \in \omega_2 \end{cases} \tag{133}$$

The cost function is non-negative. Indeed, since the sum is over misclassified points, if $\mathbf{x}_m \in \omega_1(\omega_2)$ then $\theta^T \mathbf{x}_m < (>) 0$ rendering the product $-y_n \theta^T \mathbf{x}_n > 0$. The cost function becomes zero, if there are no misclassified points, i.e. $\mathcal{Y} = \emptyset$, which corresponds to a solution. The perceptron cost function is not differentiable at all points. It is a continuous piece-wise linear function. Indeed, let us write it in a slightly different way,

$$J(\theta) = \left(-\sum_{n: \mathbf{x}_n \in \mathcal{Y}} y_n \mathbf{x}_n^T \right) \theta \tag{134}$$

This is a linear function with respect to θ, as long as the number of misclassified points remains the same. However, as one slowly changes the value of θ which corresponds to a change of the (direction/position of the hyperplane), there will be a point where the number of misclassified samples in $\mathcal{Y}$ suddenly changes; this is the time, where a sample changes its relative position with respect to the (moving) hyperplane. Hence, the set $\mathcal{Y}$ is modified. After this change, $J(\theta)$ will correspond to a new linear function.

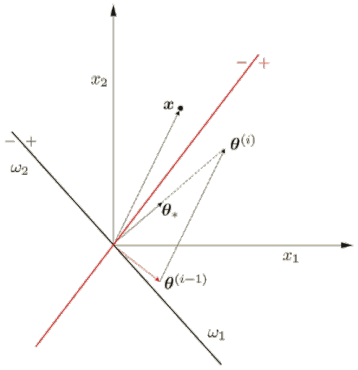
The Perceptron Algorithm: It can be shown that, starting from an arbitrary, $\boldsymbol{\theta}^{(0)}$, the following iterative update,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \sum_{n:\mathbf{x}_n \in \mathcal{Y}} y_n \mathbf{x}_n \tag{135}$$

converges after a finite number of steps. The parameter μ_i is the user-defined step-size, judicially chosen to guarantee convergence. Besides the previous scheme, another version of the algorithm considers one sample per iteration in a cyclic fashion, till the algorithm converges. Let us denote by $y_{(i)}, \mathbf{x}_{(i)}, (i) \in \{1, 2, ..., N\}$, the training pair that is presented in the algorithms at the ith iteration step. In the pattern by pattern perceptron algorithm, the algorithm becomes:

$$\boldsymbol{\theta}^{(i)} = \begin{cases} \boldsymbol{\theta}^{(i-1)} + \mu_i y_{(i)} \mathbf{x}_{(i)}, & \text{if } \mathbf{x}_{(i)} \text{ is misclassifed by } \boldsymbol{\theta}^{(i-1)}, \\ \boldsymbol{\theta}^{(i-1)}, & \text{otherwise} \end{cases} \tag{136}$$

In other words, starting from an initial estimate, e.g., taken to be equal to zero, $\boldsymbol{\theta}^{(0)}$, we test each one of the samples, $\mathbf{x}_n, n = 1, 2, ..., N$. Once all samples have been considered, we say that one epoch has been completed. If no convergence has been attained, all samples are reconsidered in a second epoch and so on. The previous algorithm is known as pattern-by-pattern or online mode of operation. Note that, the term "online" here indicates that the total number of data samples is fixed and the algorithm considers them in a cyclic fashion, epoch after epoch. After a successive finite number of epochs, the algorithm is guaranteed to converge. Note that for convergence, the sequence μ_i must be appropriately chosen. For the case of the perceptron algorithm, convergence is still guaranteed even if μ_i is a positive constant, $\mu_i = \mu > 0$, usually taken to be equal to one. The following figure provides a geometric interpretation of the perceptron rule. The sample $\mathbf{x}$ is misclassified by the hyperplane, $\boldsymbol{\theta}^{(i-1)}$. Since $\mathbf{x}$ lies in the $(-)$ side of the hyperplane and it is misclassifed, it belongs to class ω_1. Hence, assuming $\mu = 1$, the applied algorithm is



$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mathbf{x} \tag{137}$$

and its effect is to turn the hyperplane to direction towards $\mathbf{x}$ so that to place it in the $(+)$ side of the new hyperplane, which is defined by the updated estimate $\boldsymbol{\theta}^{(i)}$. Once the perceptron algorithm has run and converged, we have available the weights, $\theta_i, i = 1, 2..., l$ of the synapses of the associated neuron/perceptron as well as the bias term θ_0. These can now be used to classify unknown patterns.

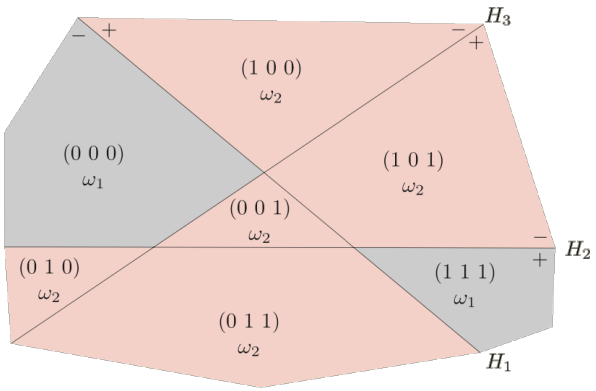
Basic Neuron Element: The features $x_0, i = 1, 2, .., l$, are applied to the input nodes. In turn, each feature is multiplied by the respective synapse (weight) and then the bias term is added on their linear combination. The outcome of this operation then goes through a nonlinear function, $f(\cdot)$, known as the activation function. In the more classical version, known as the McCulloch-Pitts neuron, the activation function is the Heaviside one, i.e.,

$$f(z) = \begin{cases} 1, & \text{if } z > 0, \\ 0 & \text{if } z \leq 0 \end{cases} \tag{138}$$

A single neuron realizes the hyperplane,

$$\theta_1 x_1 + \theta_2 x_2 + ... + \theta_l x_l + \theta_0 = 0, \tag{139}$$

in the input (feature) space. We will now see how to combine neurons, in a layer-wise fashion, in order to construct nonlinear classifiers. We will follow a simple constructive proof, which unveils certain aspects of neural networks. As a staring point, we consider classes in the feature space, which are formed by unions of polyhedral regions, as shown in the figure below,



Consider three neurons, realizing three hyperplanes, H_1, H_2, H_3, of the previous figure, respectively. The corresponding outputs, denoted as y_1, y_2, y_3 form the label of the region associated with the input pattern, which is applied on the input nodes. Indeed, if the weights of the synapses have been appropriately set, then if a pattern originates from the region, say, (010), then the first neuron will fire a zero ($y_1 = 0$), the second a one ($y_2 = 1$) and the third a zero ($y_3 = 0$). In other words, this layer of neurons forms a mapping of the input space into the 3-D (three neurons) one. We refer to this as the first hidden layer. More specifically, the mapping is performed on the vertices of the unit cube in $\mathbb{R}^3$, as shown below. The neurons of the first hidden layer perform a mapping from the input feature space to the vertices of a unit hypercube. Each region is mapped into a single vertex. Each vertex of the hypercube is now linearly separable from all the rest and can be separated by a (hyper)plane realized by a neuron. If p instead of three neurons are used, the mapping is on the vertices of the $p-$dimensional unit cube.

We will now use this new representation as input, which feeds the neurons of a second layer, which is constructed as follows. We choose all regions which belong to one class. Assume that regions (000) and (111) define class ω_1. Recall that, each of the two corresponding vertices is now linearly separable from the rest. This means that we can use a neuron/perceptron in the transformed space, which will place one vertex in the (+) side and the rest in the (-) one, as shown in the last figure. The resulting structure/network is shown in the figure in the next slide. The resulting network has a second layer of hidden neurons. The output z_1 of the left neuron will fire an "1" only if the input pattern originates from the region 000 and it will be at "0" for all other patterns. For the neuron on the right, the output z_2 will be "1" for all the patterns coming from region (111) and zero for all the rest. Note that, this second layer of neurons has performed a second mapping, this time to the unit rectangle in $\mathbb{R}^2$. This mapping provides a new representation of the input patterns, and this representation encodes information related to the classes of the regions. By successive mappings, we have transformed our originally nonlinearly separable task, to one which is linearly separable. Indeed, the point (00) can be linearly separated from (01) and (10) and this can be realized by an extra neuron operating in the (z_1, z_2) space. The latter is known as the output neuron, since it provides the final classification decision. We say that this network of neurons is a feed-forward one, since information flows in the forward direction from the input to the output layer. It comprises the input layer, which is a non-processing one, two hidden layers (the term hidden is self-explained) and one output layer. We call such a Neural Network (NN) a three layer network, without counting the input layer of non-processing nodes.

We have constructively shown that a three layer feed-forward NN can, in principle, solve any classification task whose classes are formed by union of polyhedral regions. The generalization to multiclass cases is straightforward, by employing more output neurons depending on the number of classes. Note that in some cases, one hidden layer of nodes may be sufficient. For example, this would be the case if class ω_1 was the union of (000) and (100) regions. Then these two vertices could be separated from the rest via a single plane and a second hidden layer of neurons would not be required. What we have said, so far, was a theoretical construction in order to highlight some analogies to the multilayer neural architecture of our brain and the concept of different representations of the input patterns via the various layers. In practice, when the data "live" in high dimensional spaces, there is no way of implementing neurons analytically so as to realize the hyperplanes. Furthermore, in real life, classes are not necessarily formed by union of polyhedral regions and, more important, classes do overlap. Hence, one needs to devise a training procedure based on a cost function. A feed-forward neural network (NN) consists of

a number of layers of neurons and each neuron is determined by the corresponding set of synaptic weights and its bias term. Networks with more than two hidden layers, are known as deep networks. From this point of view, a NN realizes a nonlinear parametric function, $\hat{y} = f_\theta(\mathbf{x})$, where $\boldsymbol{\theta}$ stands for all the weights present in the network. Thus, training a NN seems not to be any different than training any other parametric prediction model. All that is needed is a) a set of training samples, b) a loss function, $\mathcal{L}(y, \hat{y})$, and c) an iterative scheme, e.g., gradient descent, to perform the optimization of the associated empirical loss function,

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} \mathcal{L}(y_n, f_\theta(\mathbf{x}_n)) \tag{140}$$

A difficulty with training NNs lies in their multilayer structure that complicates the computation of the involved gradients, which are needed for the optimization. Moreover, the McCulloch-Pitts neuron involves the discontinuous Heaviside activation function, which is not differentiable. A first step in developing a practical algorithm for training a NN is to replace the Heaviside activation function with a differentiable one. One possibilitiy is to adopt the logistic sigmoid function

$$f(z) = \sigma(z) := \frac{1}{1 + \exp(-az)} \tag{141}$$

Another alternative is the hyperbolic tangent function:

$$f(z) = a \tanh\left(\frac{cz}{2}\right) \tag{142}$$

where c and a are controlling parameters.

Having adopted a differentiable activation function, we are ready to proceed with developing the gradient descent iterative scheme for the minimization of the cost function. We will formulate the task in a general framework, known as the Gradient Descent Scheme. What is more important is to grasp the rationale behind the algorithm and not the details. Let $(\mathbf{y}_n, \mathbf{x}_n), n = 1, 2..., N$, be the set of training samples. Note that we have assumed multiple output variables, assembled as a vector. We assume that the network comprises L layers; $L-1$ hidden and one output layer. Each layer consists of $k_r.r = 1, 2, ..., L$, neurons. Thus, the output vectors are:

$$\mathbf{y}_n = [y_{n1}, y_{n2}, ..., y_{nk_L}]^T \in \mathbb{R}^{k_L}, n = 1, 2, ..., N \tag{143}$$

For the sake of the mathematical derivations, we also denote the number of input nodes as k_0; i.e. $k_0 = l$, where l is the dimensionality of the input feature space. Let $\boldsymbol{\theta}_j^r$ denote the synaptic weights associated with the jth neuron in the rth layer, with $j = 1, 2, ..., k_r$, and $r = 1, 2, ..L$, where the bias term is included in $\boldsymbol{\theta}_j^r$, i.e.,

$$\boldsymbol{\theta}_j^r := [\theta_{j0}^r, \theta_{j1}^r, ..., \theta_{jk_{r-1}}^r]^T \tag{144}$$

The basic iterative step for the gradient decent scheme is written as

$$\boldsymbol{\theta}_j^r(\text{new}) = \boldsymbol{\theta}_j^r(\text{old}) + \Delta\boldsymbol{\theta}_j^r \tag{145}$$

where

$$\Delta\boldsymbol{\theta}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{\theta}_j^r}\bigg|_{\boldsymbol{\theta}_j^r(\text{old})} \tag{146}$$

The parameter μ is the user-defined step size (it can also be iteration-dependent) and J denotes the cost function. For example, if the squared error loss is adopted, we have

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} J_n(\boldsymbol{\theta}) \tag{147}$$

and

$$J_n(\boldsymbol{\theta}) = \frac{1}{2}\sum_{k=1}^{k_L}(\hat{y}_{nk} - y_{nk})^2 \tag{148}$$

where $\hat{y}_{nk}, k = 1, 2, ..., k_L$ are the estimates provided at the corresponding output nodes of the network. We will consider them as the elements of a corresponding vector, $\hat{\mathbf{y}}_n$. The main difficulty in the backpropagation algorithm lies in the computation of the gradients. Note that the output of the network relates directly to the parameters associated with the neurons of the last (output) layer. Thus, the computation of the corresponding gradients poses no problems. Business as usual. However, the output of the network is related indirectly with the parameters of the neurons comprising the hidden layers. This is because the outputs/responses of the hidden layers are transformed by the neurons of the layers above. The closer to the input is a layer, the more transformations the respected neuron responses undergo, as they propagate through the layers higher in the hierarchy. To compute the gradients, two types of computations are performed:

- Forward Computations: For a given input, $\mathbf{x}_n$, employ the currently available estimates of the parameters and compute the output of the network, say, $\hat{y}_n$, which depends on the current estimates.
- Backward Computations: Using the desired response, y_n, and the predicted one, $\hat{y}_n$, compute the corresponding gradients of the cost function w.r. to all the parameters. To this end, the computations propagate backwards: (1) Compute the gradients of the parameters of the neurons of the last layer L. (2) Using the previously computed gradients and the chain rule of derivation (to account for the imposed, by the network, transformations), compute the gradients of the parameters of the neurons of layer $L-1$. (3) The above procedure caries on, backwards, till all the gradients, including those in the first hidden layer, have been computed.

Computation of the gradients: Let z_{nj}^r denote the output of the linear combiner of the jth neuron in the rth layer at time instant n, when the pattern $\mathbf{x}_n$ appears at the input nodes. Then, we can write that

$$\sum_{m=1}^{k_{r-1}} \theta_{jm}^r y_{nm}^{r-1} + \theta_{j0}^r = \sum_{m=0}^{k_{r-1}} \theta_{jm}^r y_{nm}^{r-1} + \theta_{j0}^r = \boldsymbol{\theta}_j^{rT}\mathbf{y}_n^{r-1} \tag{149}$$

where by definition

$$\mathbf{y}_n^{r-1} := [1, y_{n1}^{r-1}, ..., y_{nk_{r-1}}^{r-1}]^T \tag{150}$$

and $y_{n0}^r \equiv 1, \forall r, n$. For the neurons at the output layer, $r = L, y_{nm}^L = \hat{y}_{nm}, m = 1, 2, .., k_L$, and for $r = 1$, we have $y_{nm}^0 = x_{nm}, m = 1, 2, ..., k_0$; that is, y_{nm}^0 are set equal to the input feature values. Hence we can now write that

$$\frac{\partial J_n}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r}\frac{\partial z_{nj}^r}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r}\mathbf{y}_n^{r-1}, \quad \text{and } \delta_{nj}^r := \frac{\partial J_n}{\partial z_{nj}^r} \tag{151}$$

Then we have

$$\Delta\boldsymbol{\theta}_j^r = -\mu\sum_{n=1}^{N}\delta_{nj}^r\mathbf{y} + n^{r-1}, r = 1, 2, ..., L \tag{152}$$

Computation of δ_{nj}^r: Here is where the heart of the backpropagation algorithm beats. For the computation of the gradients, δ_{nj}^r, one starts at the last layer, $r = L$, and proceeds backwards towards $r = 1$; this philosophy justifies the name given to the algorithm. At $r = L$, we have that

$$\delta_{nj}^L = \frac{\partial J}{\partial z_{nj}^L} \tag{153}$$

For the squared error loss function,

$$J_n = \frac{1}{2}\sum_{k=1}^{k_L}(f(z_{nk}^L) - y_{nk})^2 \tag{154}$$

Hence

$$\delta_{nj}^{L} = (\hat{y}_{nj} - y_{nj}) f'(z_{nj}^{L}) = e_{nj} f'(z_{nj}^{L}), j = 1, 2, .., k_L \tag{155}$$

where $f'(\cdot)$ denotes the derivative of $f(\cdot)$, and e_{nj} is the error associated with the jth output variable at time n. Note that for the last layer, the computation of the gradient is straightforward. For the case $r < L$, due the to success dependence between the layers, the value of z_{nj}^{r-1} influences all the values $z_{nk}^{r}, k = 1, 2, ..., k_r$ of the next layer. Employing the chain rule for different, we get

$$\delta_{nj}^{r-1} = \frac{\partial J_n}{\partial z_{nj}^{r-1}} = \sum_{k=1}^{k_r} \frac{\partial J_n}{\partial z_{nk}^{r}} \frac{\partial z_{nk}^{r}}{\partial z_{nj}^{r-1}} \tag{156}$$

or

$$\frac{\partial J_n}{\partial z_{nj}^{r-1}} = \sum_{k=1}^{k_r} \delta_{nk}^{r} \frac{\partial z_{nk}^{r}}{\partial z_{nj}^{r-1}} \tag{157}$$

However,

$$\frac{\partial z_{nk}^{r}}{\partial z_{nj}^{r-1}} = \frac{\partial \big(\sum_{m=0}^{k_{r-1}} \theta_{km}^{r} y_{nm}^{r-1} \big)}{\partial z_{nj}^{r-1}}, \quad \text{where } y_{nm}^{r-1} = f(z_{nm}^{r-1}) \tag{158}$$

which leads to

$$\frac{\partial z_{nk}^{r}}{\partial z_{nj}^{r-1}} = \theta_{kj}^{r} f'(z_{nj}^{r-1}) \tag{159}$$

and combining with our expression from above, we obtain for $j = 1, 2, .., k_{r-1}$

$$\delta_{nj}^{r-1} = \left(\sum_{k=1}^{k_r} \delta_{nk}^{r} \theta_{kj}^{r} \right) f'(z_{nj}^{r-1}) := \delta_{nj}^{r-1} = e_{nj}^{r-1} f'(z_{nj}^{r-1}) \tag{160}$$

Some remarks on the backpropagation algorithm. One possibility to terminate the algorithm is to track the value of the cost function, and stop the algorithm when this gets smaller than a preselected threshold. An alternative path is to check for the gradient values and stop when these become small. As it is the case with all gradient descent schemes, the choice of the step size, μ, is very critical; it has to be small to guarantee convergence, but not too small, otherwise convergence speed slows down. Adaptive values of μ, whose value depends on the iteration are more appropriate. Soon, such techniques will be discussed. Due to the highly nonlinear nature of the NN task, the cost function in the parameter space is, in general, of a complicated form and there exist local minima, where the algorithm can be trapped. If such a local minimum is deep enough, the obtained solution can be acceptable. However, this may not be the case and the solution can be trapped in a shallow minimum resulting in a bad solution. However, this "shallow minima" view has been challenged in the context of deep architectures. As we will discuss soon, in the case of networks with many layers, shallow minima may not necessarily be a major problem. Saddle points become the critical issue. In practice, random initialization of the weights is carried out. Yet, initialization remains a critical part of the algorithm.

Pattern-by-pattern optimization: The previous scheme is of the batch type of operation, where the weights are updated once per epoch. The alternative route is the pattern-by-pattern/online mode of operation; the weights are updated at every time instant when a new pattern appears in the input.

Mini-batch operation: There are also intermediate ways, where the update is performed every $N_1 < N$ samples; this technique is also referred as mini-batch mode of operation. Batch and mini-batch modes have an averaging effect on the the computation of the gradients.

Due to the hierarchical computations of the gradients, it turns out that their computation involves a sequence of products of parameters with derivatives of the activation function. The closer to the input layer we are, the more products the computation of the respected gradients involve. Taking into account that the derivatives of the activation function can be less than one (e.g., for sigmoid functions can be very small), and if the parameters values are not very large, this can make the gradients, associated to the parameters in the lower layers, vanishingly small, especially if networks with many layers are involved. This can make learning extremely slow. On the

other extreme, if the values of the parameter estimates happen to take large values, this may lead the values of the gradients to explode. As a result, this can disturb the learning process, by pushing the estimates to wrong regions in the parameters' space. Another related problem is that gradients in different layers can take values of different scales. Thus, some layers can learn faster that others, and this can make the learning process unstable. To cope with such difficulties, a number of modifications of the basic gradient scheme and a number of practical hints have been proposed.

Gradient descent with a momentum term: One way to improve the convergence rate is to employ the so called momentum term, a. The correction term is now modified as

$$\Delta\boldsymbol{\theta}_j^r(\text{new}) = a\Delta\boldsymbol{\theta}_j^r(\text{old}) + \Delta\boldsymbol{\theta}_j^r \tag{161}$$

The effect is to increase the step size in regions, where the cost function exhibits low curvature. Indeed, assume that the gradient is approximately constant over a number of steps, say I. Then, it can be shown that

$$\Delta\boldsymbol{\theta}_j^r(I) \approx -\frac{\mu}{1-\alpha}\mathbf{h} \tag{162}$$

where $\mathbf{g}$ is the gradient over the I steps. That is, the use of the momentum term increases the correction by a factor $1-\alpha$. Note that adaptive versions for the momentum term a are possible and popular.

A number of alternatives have been proposed, and the topic of speeding up the convergence of the backpropagation algorithm has been a hot topic of research, and many variants have been proposed over the years. For example: Newton-type and related simplified versions for computing the associated Hessian matrix. A number of versions, using more recent results on optimization, have also been suggested; for example, schemes based on the ADAGARD or on the Nesterov rationale.

The choice of a loss function for the optimization is tightly related with the choice of the output activation function. A wrong combination can severely affect the learning performance of a network. Let us select the squared error loss function and the logistic sigmoid function as the output nonlinearity, i.e.,

$$f(z) = \sigma(z) = \frac{1}{1+\exp(-az)}, \quad J(e) = \frac{1}{2}(y-\hat{y})^2 \tag{163}$$

where a single output neuron is considered and the time index has been suppressed. Assume L layers and let the vector of the parameters, associated with the single output neuron, be $\boldsymbol{\theta}^l$. The vector outputs of the previous $(L-1)$ layer is denoted as $\mathbf{y}^L := [y_1^{L-1}, y_2^{L-1}, ..., y_{k_{L-1}}^{L-1}]$. Then, the output of the network will be:

$$\hat{y} = \sigma(z^L), \quad z^L := \boldsymbol{\theta}^{L^T}\mathbf{y}^{L-1} \tag{164}$$

where the bias term has been included in the vector of parameters. For the specific combination of loss and activation functions, it turns out that

$$\frac{\partial J}{\partial \boldsymbol{\theta}^L} = (y-\hat{y})\sigma'(z^L)\mathbf{y}^{L-1} \tag{165}$$

Observe that for values of z^L not close to zero, the derivative of the logistic sigmoid function takes very small values, due to its saturating nature. However, very small values of the gradient lead to considerable slow down of the convergence of the gradient descent type algorithms. In contrast, this is not the case, if the squared error loss function is combined with a linear activation function. This is a perfectly good combination. If one adopts as target values, in the classification task, the 0,1 values, i.e., $y_n \in \{0,1\}$, and assuming k_L output nodes, the **cross-entropy** cost is defined as:

$$J = -\sum_{n=1}^{N}\sum_{k=1}^{k_L}(y_{nk}\ln\hat{y}_{nk} + (1-y_{nk})\ln(1-\hat{y}_{nk})) \tag{166}$$

where N is the number of training points. The minimum of this cost function is achieved when $y_{nk} = \hat{y}_{nk}$. View $\hat{y}_{nk}$ as the probability of observing a "1" at the respective node, then the probability $P(\mathbf{y}_n)$ is equal to:

$$P(\mathbf{y}_n) = \prod_{k=1}^{k_L} (\hat{y}_{nk})^{y_{nk}} (1 - \hat{y}_{nk})^{1-y_{nk}} \tag{167}$$

Thus, the cross entropy can be interpreted as the negative log-likelihood function over the training samples. It turns out that, combining the cross entropy with the logistic sigmoid activation in the output nodes renders the associated gradients independent of the respective derivative and the gradients depend solely on the errors committed.

We desire a probabilistic interpretation of the cost function, which results in another activation function, known as the softamx.

Softmax activation function: Although we have interpreted the outputs as probabilities, there is no guarantee that these add to one. This can be enforced if the activation function takes the form

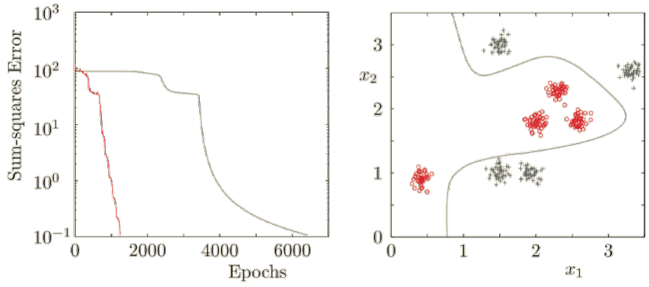
$$\hat{y}_{nk} = \frac{\exp(z_{nk}^l)}{\sum_{m=1}^{k_L} \exp(z_{nk}^l)} \tag{168}$$

which is known as the softmax function. It turns out that, combining the softmax activation with the cross-entropy loss makes the gradients equal to

$$\frac{\partial J}{\partial \boldsymbol{\theta}^L} = (y - \hat{y})\mathbf{y}^{L-1} \tag{169}$$

where time and node indices have been suppressed. Thus, the gradients depend on the error and no derivative is involved.

Example - Deep Learning



The classification task consists of two classes, each being the union of four regions. Each region consists of normally distributed random vectors. A total of 400 training vectors were generated, 50 from each distribution. A multilayer perceptron with three neurons in the first and two neurons in the second hidden layer were used, with a single output neuron. The activation function was the logistic one with $a = 1$ and the desired outputs 1 and 0, respectively, for the two classes. The momentum and the adaptive momentum algorithms, as explained in the book, were used. The weights were initialized by a uniform pseudorandom distribution between 0 and 1. The obtained results are shown in the figure on the previous page.

The Rectified Linear Unit (ReLU) Besides the two already mentioned activation functions, more recently, a new one has become very popular for use in the hidden layers, especially in the context of deep networks. The rectified linear unit (ReLU) is defined as

$$f(z) := \max(0, z) \tag{170}$$

It has been established, by now, that the use of the ReLU in the context of deep networks, with many layers, can improve the training time significantly. Observe that the ReLU has derivatives that their values remain large for large positive values of z; That is, in the region where the neuron remains active. Thus, it is advisable to initialize the respective biases to some positive small value, e.g.,$\theta_0 = 0.1$; this increases the probability the the input to the activation has positive values. Note that at $z = 0$, the derivative is not defined; yet, in the extreme case that z is exactly zero, one can set the derivative either equal to zero or to one. The major disadvantage of the ReLU is that learning is freezing when $z < 0$. To bypass this obstacle, a number of variants have been proposed. Consider the function

$$f(z) = \max(0, z) + \alpha \min(0, z) \tag{171}$$

When $\alpha = -1$, the resulting is known as the absolute value rectification. When α is assigned a fixed small value, e.g., $\alpha = 0.01$, the resulting function is coined as the leaky ReLU. When α is left as a parameter to be learned during the training, it is known as the parametric ReLU. Maxout unit: In this variant, a fixed number of, say k, different ReLUs are employed, which are learned during the training. The output of the neuron is selected as the maximum one among the k ones.

Which nonlinearity is the best? Unfortunately, there is not a universal answer to that. It depends on the data and the problem at hand. At the time of developing these slides, it seems that the ReLU versions are the preferable choice for the hidden layers, in a number of mainstream applications.

Regularization: A crucial factor in training NNs is to decide the size of the network. The size is directly related to the number of weights to be estimated and we know that, in any parametric modeling method, if the number of free parameters is large enough with respect to the number of training data, overfitting is bound to happen. Regularizing the cost function is a way to cope with the problem.

Pruning a Network: Often, in practice, one starts with a large enough number of neurons and then use a regularization technique to push the less informative weights to low values, which are then removed. This is known as pruning. There exists a number of different regularization approaches, which have been proposed and used over the years. In general, we addressing the problem of modular selection.

Model Order Selection: For linear regression, the output is a function of weighted components and use the least squares method. The optimal number of components can be determined using something known as Mallow's statistic. In the case of time series,

$$x[n] = \sum_{j=1}^{p} \theta_j x[n-j] + w[n] \tag{172}$$

where $w[n]$ is 0 mean white noise with variance ρ. For this problem, suppose with have N samples, which we use to estimate all θ's and ρ. Let $\hat{\theta}$ be the estimates of θ for $s = 1, ..., p$ and $\hat{\rho}$ be the estimate of ρ_k, where we have $k = 1, .., M$ models. We next calculate Akaike's information criterion (AIC), which is:

$$AIC_k = N \ln \hat{\rho}_k + 2m_k \tag{173}$$

Here m_k is the number of parameters. We choose the model with the lowest value of AIC_k. In 1977, Kashyap et. al proposed a Bayesian Information criterion. We calculate BIC for the kth model:

$$BIC_k = N \ln \hat{\rho}_k + m_k \ln N \tag{174}$$

where m_k is the number of parameters in the kth model and $\hat{\rho}_k$ is the variance of the kth fitted model. In this case, we have compound hypothesis testing problem (the model order and model itself are unknown). Effectively, selecting a good model architecture is a basic problem. Neural networks are non-linear models and have many heuristics.

Weight decay: This path refers to a typical cost function regularization via the Euclidean norm of the weights. Instead of minimizing a cost function, $J(\boldsymbol{\theta})$, its regularized version is used, i.e.,

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda ||\theta||^2 \tag{175}$$

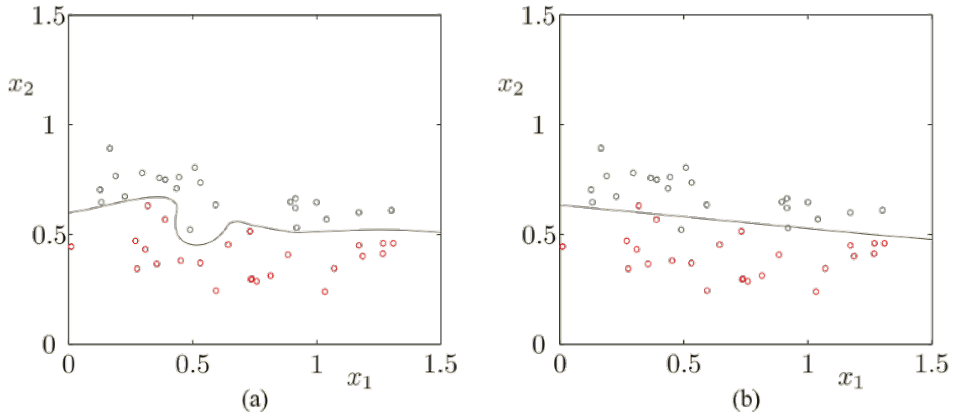
In general, it is not a good practice to include the bias terms in the norm. As it is the case with the ridge regression task, this affects the translation invariant properties of the network. Moreover, it is even better if one groups the parameters of different layers together and employs different regularizing constants for each group. More recently, the use of the sparsity promoting l_1 norm has been proposed in places of the Euclidean norm.

Weight Elimination: Another path is to include other functions than the previous norms. An example is,

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \sum_{k=1}^{K} \frac{\theta_k^2}{\theta_h^2 + \theta_k^2} \tag{176}$$

where K is the number of weights involved and θ_h is a pre-selected threshold. A careful look at this function reveals that, if $\theta_k < \theta_h$, the penalty term goes to zero very fast. In contrast, for values $\theta_k > \theta_h$, the penalty term

tends to unity. In this way, less significant weights are pushed towards to zero. The samples of the two classes are denoted by black and red circles respectively. Figure (a) on the left corresponds to a multilayer perceptron with two hidden layers and 20 neurons in each of them, amounting to a total of 480 weights. Training was performed via the backpropagation algorithm. The overfitting nature of the resulting curve is readily observed. Figure (b) on the right corresponds to the same multilayer perceptron trained with a pruning algorithm. Finally, only 25 of the 480 weights have survived, and the curve is simplified to a straight line.



Adding some small noise to the input data turns out to be equivalent with modifying the cost function by adding an extra term, which acts as a regularizer. Adding some small noise to the unknown parameters, during their training, pertubes the solution. Using Taylor series expansion arguments around this perturbation, it turns out that this procedure is equivalent with regularizing the cost function via the norm of the gradient of the w.r. to the parameters. However, recall that all inference models in statistics for Gaussian noise are for linear systems; neural networks are non-linear. The two major problems with neural networks inference model development are: (1) hierarchial structure and (2) non-linearity. The source of overfitting is the relatively limited number of the training data compared to the size of the network. Thus, increasing the data size has an equivalent effect as regularization. It decreases overfitting. In certain applications, one can artificially generate more data. For example, in an OCR task, one can generate many characters in different rotations. Similar arguments hold for object recognition tasks. Soon, we will discuss ways of generating fake data.

Dropout This is a more recent technique that deals with overfitting, in the context of deep networks. The term dropout refers to dropping out/removing neurons and/or input nodes in a neural network. One starts with a large enough network, comprising, say, K nodes. Choose a training algorithm, e.g., any version of the gradient descent backpropagation algorithm. At each iteration step of the algorithm: Retain each node (hidden or input), together with its incoming and outgoing connections, with probability P. Thus, each node may be removed with probability $1 - P$. Train the remaining nodes according to the selected algorithm. Thus, at each iteration step, a different subnetwork is trained. In other words, at each iteration step, only the parameters of one subnetwork are updated. The parameters of the "removed" nodes are left unchanged, frozen at their current estimates. Once training has been completed and convergence has been achieved, during the test phase, each parameter is multiplied by the probability P. At each iteration step, a different subnetwork is trained. This can be thought of as being equivalent of training a large number of networks (theoretically 2^K). Once training is over, one combines the trained subnetworks by an averaging rationale. This reminds of the bagging approach to combine predictors. Yet, it is different. In dropout, there is a large overlap and parameter sharing among the different subnetworks. Typical values for the probabilities are: For hidden layers $P = 0.5$ and for input units $P = 0.8$. A heuristic explanation on why this technique works is that it reduces co-adaptations of neurons, since at each iteration different neurons are updated. Thus, the network is forced to learn more robust features, that are useful in conjunction with the different subnetworks. In other words, the network is forced to learn while parts of the network are missing at random. Some more theoretically pleasing arguments have been given via a Bayesian inference view of a neural network.

So far, we focused on how to train neural networks so that to learn a specific input-output mapping. Our interest now turns on what a neural network is capable to learn? To this end, some strong theoretical results have been

developed and are still being developed. Let us consider a two-layer network, with one hidden layer and with a single output linear node. The output of the network is then written as

$$\hat{g}(\mathbf{x}) = \sum_{k=1}^{K} \theta_k^{\circ} f(\boldsymbol{\theta}_k^{hT} \mathbf{x}) + \theta_0^{\circ} \tag{177}$$

where $\boldsymbol{\theta}_k^h$ denotes the synaptic weights and bias term defining the kth neuron and superscript "o" refers to the output neuron. Then, the following theorem holds true:

Theorem: Get $g(\mathbf{x})$ be a continuous function defined in a compact (closed and bounded) subset $S \subset \mathbb{R}^l$ and any $\epsilon > 0$. Then there exists a $K(\epsilon)$ and a two-layer network of the previous so that

$$|g(\mathbf{x}) - \hat{g}(\mathbf{x})| < \epsilon, \quad \forall \mathbf{x} \in S \tag{178}$$

It has been shown that the approximate error decreases according to an $O\left(\frac{1}{K}\right)$ rule. In other words, the input dimensionality does not enter into the scene and the error depends on the number of neurons used. The theorem states that a two-layer NN network is sufficient to approximate any continuous function. However, what the theorem does not say is how big such a network should be, in terms of the required number of neurons in the single layer. It may be that a very large number of neurons is needed in order to obtain a good enough approximation. This is where the use of more layers can be advantageous. Using more layers, the overall number of neurons, needed to achieve certain approximation, may be much smaller. This is point that ignites our the interest for deep networks, involving many hidden layers.

We have already discussed that each layer of a neural network provides a different description of the input patterns. In the context of our previous presentation: The input layer described each pattern as a point in the feature space. The first hidden layer of nodes formed a partition of the input space and placed each input point in one of the regions, using a coding scheme of zeros and ones at the outputs of the respective neurons. This can be considered as a more abstract representation of our input patterns. The second hidden layer of nodes, based on the information provided by the previous layer, encoded information related to the classes; this is a further representation abstraction, which carries some type of semantic meaning. For example, it could provide information of whether a tumor is malignant or benign, in a related a medical application. It turns out that, the previous reported hierarchical type of representations of the input patterns mimics the way that a mammal's brain follows in order to understand and sense the world around us; in the case of humans, this is the physical mechanism in the brain, which intelligence is built upon. The brain of the mammals is organized in a number of layers of neurons and each layer provides a different representation of the input percept. In this way, different levels of abstraction are formed, via a hierarchy of transformations.

For example, in the primate visual system, this hierarchy involves first detection of edges, then formation of primitive shapes and every subsequent stage forms more complex visual shapes, till finally a semantics concept is formed; e.g., a car moving in a video scene, a person sitting in an image. The cortex of our brain can be seen as a multilayer architecture with 5-10 layers dedicated only to our visual system. An issue that is now raised is whether one can obtain an equivalent input-output representation via a relatively simple functional formulation, e.g., via networks with less than three layers of neurons/processing elements, maybe at the expense of more elements per layer.

Using networks with more layers can lead to more compact representations of the input-output relation. Results from the theory of circuits of Boolean functions suggest that a function, which can compactly be realized by, say, k layers of logic elements, may need an exponentially large number of elements if it is realized via $k-1$ layers. Some of these results have been generalized and are valid for learning algorithms in some special cases. For example, it has been shown that, for a class of deep networks and target functions, one needs a substantially smaller number of nodes to achieve a predefined accuracy compared to a shallow one. A major drawback of multilayer NNs is that their training can become difficult. This drawback becomes more severe if more than two hidden layers are used. The more layers one uses, the more difficult the training becomes. Historically, in the 1990's, the effort to train large networks was, practically, abandoned. For a long time, it was believed that, this was due to the existence of many local minima, which caused the learning algorithm to be trapped in a shallow one. To remedy such a drawback, the algorithm was randomly initialized from different points a number of times, hoping for the best result.

The view point concerning local minima is now challenged, as new results started coming out around 2015. Theoretical as well as experimental evidence point out that the major drawback lies not in the local minima but in the saddle points. This is an ongoing and active research area. Under some simplifications, it has been shown that in large size networks most local minima yield low cost function values and result to similar performance. Moreover, the probability of finding a poor local minimum decreases fast as the size of the network increases ([Choromanska, et.al. 2015]). In high dimensional spaces, the major drawback seems to be posed by the proliferation of the saddle points. The existence of such points can slow down the convergence of the training algorithms dramatically (Dauphin, et.al, 2014]). Although the exact effect of these findings on the gradient-type algorithms is not yet clear, it seems that they are finally able to escape such critical points, in spite of the very small values of the corresponding gradients ([Goodfellow, et. al. 2015]). A particularly interesting result has been derived in [Xie, et. al., 2017]. Focusing on a single hidden layer network, involving ReLU activations, they proved that, in spite of the nonconvexity of the cost function: Under certain assumptions, there are no spurious local minima points. If the squared norm of the gradient matrix is bounded by an ϵ, then the (squared) error on the training set is also bounded by $O(\epsilon)$. The generalization error is bounded by $O\big(\epsilon + \frac{1}{\sqrt{N}}\big)$, where N is the number of training data points.

The previous result confirms what is known and observed in practice. Neural networks can perform well even without regularization, provided that they are trained with lots of training points. Of course, regularization improves the performance. Currently, the success of the neural networks seems to lie in the available computational power combined with the availability of large training data sets. The combination of ReLU activation functions with the dropout technique, together with some practical hints, concerning initialization, seem to offer the secret of their success. The use of appropriate pre-training techniques, as we will soon see, can also be beneficial in certain cases.

A major class of NNs, known as convolutional neural networks, employ convolutions instead of multiply-add type of neurons. We will first review the "why" behind the convolutions. The input to any classifier/learner is presented with a set of features. Each input vector, x_n, in the training set is a point in the feature space. The features should encode, in a compact way, information that resides in the raw/sensed data and it is related to the learning task at hand. If, instead, the input to a neural network were the pixels of a 256×256 image, this would correspond to a vector in a space of dimension equal to 65536. If the first hidden layer had, say, 20000 nodes, this would amount to approximately 1.3 billion synapses! Adding more layers, this number would explode further. One among the most popular ways to generate features from an image (and not only) has traditionally being to "run" a filter over the image. Filtering exploits underlying correlations/relations among the pixels; different filters can extract different type of information.

Convolution: In the current context, filtering will be viewed as a cross-correlation operation between the filer matrix, known as the kernel matrix, H, and the image array, I. The output matrix, O, is known as a feature map. Let us assume for simplicity, that

$$H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \quad \text{and} \quad I = \begin{bmatrix} I(1,1) & I(1,2) & I(1,3) \\ I(2,1) & I(2,2) & I(2,3) \\ I(3,1) & I(3,2) & I(3,3) \end{bmatrix} \tag{179}$$

The convolution between kernel matrix, H, and the image, I, will be the 2×2 feature map array, O, with elements

$$O(n,m) = \sum_{i=1}^{2} \sum_{j=1}^{2} h_{ij} I(n+i-1, m+j-1), n, m = 1, 2 \tag{180}$$

A breakthrough in training neural networks came in the late 1980s (Le Cun and co-workers), when the feature generation step, via convolutions, was integrated as part of a neural network. Instead of using fixed kernel matrices, it was left to the network to learn the elements of the kernel matrix as part of the training process. Thus, the first layers of a neural network were dedicated to perform convolutions instead of simple multiply-add operations. These constitute the layers where the features are learned from the input raw data and feed subsequent layers of the NN. The basic steps performed in the front end convolution layers are: (1) The convolution step, (2) The nonlinearity step, and (3) The pooling step. The first layer in a CNN comprises the parameters of the kernel matrix. If the input nodes correspond to the (raw) pixels of an image array, the output of the convolutional layer

is the corresponding feature map array. Thus, the parameters comprise the kernel array and they are shared among the input pixels; moreover, in place of the multiply-add operations convolutions are performed, instead. However, instead of a single kernel matrix, multiple ones are used; each one is expected to extract different type of information, to be encoded via a different feature map array. In the figure, three such kernel arrays are shown to "scan" the input image, searching for "hidden information". There is strong evidence from the visual neuroscience that, similar computations are performed in the human brain. The idea of employing convolutions was first exploited in the neogognitron ([Fukushima]).

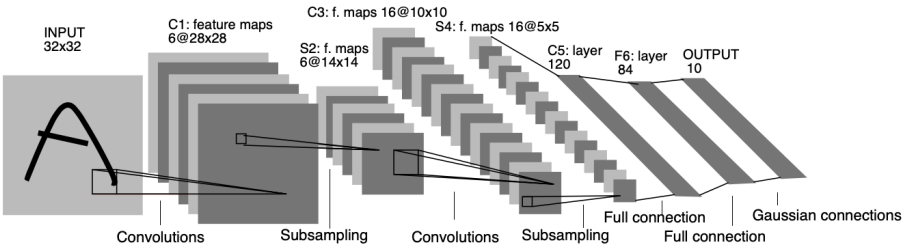
A welcome byproduct of the convolution step is that in this way, the network becomes invariant to translations (i.e. translation invaraince). The same kernel matrix is slided all over the input image array. Thus, if an object has been moved within an image, the only difference is that, in the feature map, the corresponding activity will move by the same amount of pixels. Each pixel in the feature map array receives input from within a specific region of the previous (input) layer.This is known as the corresponding receptive field. The depth refers to the number of kernel matrices (filters) that are employed. For each filter, a corresponding feature map image array results. Stride is the number of pixels by which one slides the filter matrix over the input matrix. When the stride is one, then we move the filters one pixel at a time. When the stride is two, then the filter jumps two pixels at a time as we slide them around. The larger the stride is the smaller the resulting feature maps become. is the number of pixels by which one slides the filter matrix over the input matrix. When the stride is one, then we move the filters one pixel at a time. When the stride is two, then the filter jumps two pixels at a time as we slide them around. The larger the stride is the smaller the resulting feature maps become. Sometimes, we pad the input matrix with zeros around the border pixels, so that we can apply the filter to the bordering elements of the input image matrix.

Once the convolution step has been completed and feature maps have been produced, a nonlinearity is applied to each pixel/element of each feature map array. Typical nonlinearities used are the sigmoid functions or the ReLUs. The latter seem to be the preferable choice currently. Note that after convolution, some of the matrix elements can become negative. These are set equal to zero, after, e.g., the application of the ReLU.

Pooling reduces the size of the feature map. To this end, one slides a window, e.g, 2×2, over the feature map, and for each location of the window a single value is selected. This is a downsampling operation. Pooling is also contributing in building into the network shift invariance properties [Bruna, et. al. 2013]. There are different scenarios. In the max pooling, the maximum value is selected. In the average pooling, the average value is selected. Other variants do, also, exist. The three stages discussed before, i.e, the convolution, the nonlinearity and the pooling steps, comprise a single layer of a convolutional network. In practice, a CNN comprises a series of such convolution layers. The first one is presented with the input image array. The second one receives as inputs the pooled features maps of the previous layer, and so on. Networks with 20-25 layers have been reported in practical applications. Finally, the feature map arrays of the last convolution layer are provided as inputs to a classifier. A softmax output NN is a popular choice, yet SVMs or other predictors can also be employed. Training of the full CNN takes place via a modified backpropagation algorithm. The modification is due to the weight sharing of the convolutional layers.

Gradient-Based Learning Applied to Document Recognition - Lecun et. al

Previously, in machine vision, the field relied on hand-designed feature extractors prior, in which accuracy is determined by the designer of the feature extractor. Lecun et. al developed one of the first automatic methods based purely on the data. We have inputs, outputs, gradients, and a loss function $Y^p = F(Z^p, W)$ as in the current machine learning framework. We are given samples $\{(Z_1, D_1), ..., (Z^P, D^P)\}$. Lecun called this gradient based learning as he updated the weights with $W_k = W_{k-1} - \epsilon(\partial E^{P_k}(W)/\partial W)$. Lecun also mentioned that the local minima are not a problem. To employ this algorithm, he used gradient back propagation (first developed by Larry Ho in 1969). LeCun first implemented what was known as LeNet-5, shown below:



The first layer has 6 feature maps, followed by a subsampling layer. The third layer has 16 features followed by a subsampling layer. The nonlinear activating function was a sigmoidal function. Layers take information from contiguous subsets of 3, followed by subsets of 4, and so on. Of importance, the network architecture is technically hand-crafted, though the individual convolutions are not. The L2 loss was used. The paper then compared the accuracy with SVM, K nearest neighbor, etc.

Stochastic Gradient Descent

Consider the weights $\theta \in \mathbb{R}^D$. Let $J(\theta)$ be the network across all $\theta = \boldsymbol{\theta}$ for all the training data. The gradient is $\nabla_\theta J(\boldsymbol{\theta})$. The gradient descent given the gradients and θ states that we update the weights with:

$$\theta^{i+1} = \theta^i - \eta \nabla_\theta J(\boldsymbol{\theta}) \tag{181}$$

However, using all the training data significantly slows the speed of training. Furthermore, we are affected any redundancies in the input data. The method of stochastic gradient descent performs the same operation for a single point in the traninig data:

$$\theta^{i+1} = \theta^i - \eta \nabla_\theta J(\theta, x^{(j)}, y^{(j)}) \tag{182}$$

Compared to regular gradient descent, where we update the weights with the entire batch of training data, SGD only considers one data point. Though the redundancy problem is reduced, $\boldsymbol{\theta}$ is more oscillatory. Mini-batch gradient descent takes advantages from both algorithms and updates for every batch of n training samples. In general n ranges from 50-256. Challenges with implementation include assigning a proper learning rate, providing a schedule for the learning rate, and annealing, which involves getting trapped in a local minima.

Momentum. Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \\ \theta &= \theta - v_t \end{aligned} \tag{183}$$

Here γ is the momentum and is usually in the range of 0.9. Momentum accelerates SGD and dampens oscillations.

Nesterov accelerated gradient. The Nesterov accelerated gradient (NAG) provides information regarding the next position of the parameters, or a rough idea of where the parameters are going to be. We can now effectively look ahead by calculating the gradient not w.r.t. to our current parameters θ but the approximate future position of our parameters:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{184}$$

Adagrad. Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data. Let θ_i be the ith parameter. Let $g_{t,i}$ be the gradient of the objective function with respect to the parameter θ_i.

$$\begin{aligned} g_{t,i} &= \nabla_{\theta_i} J(\theta_{t,i}) \\ \theta_{t+1,i} &= \theta_{t,i} - \eta g_{t,i} \end{aligned} \tag{185}$$

In this update rule, Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{186}$$

Here $G_{t,ii}$ is the L2 norm and $\epsilon \neq 0$ so as to avoid a 0 in the denominator.

Adam (Adaptive Moment Estimation). Adam is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{187}$$

m_t and v_t are the estimates of the first and second moments of the gradients:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \tag{188}$$

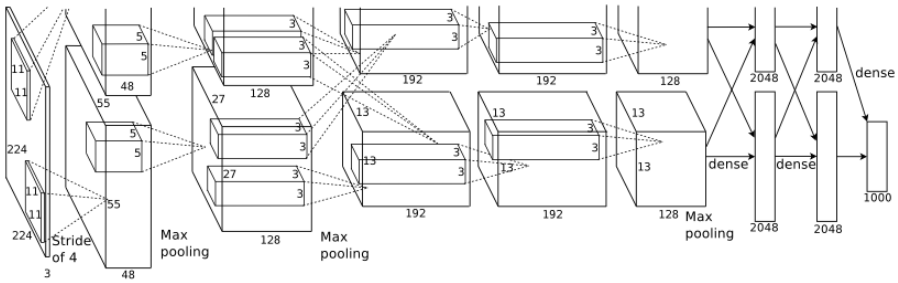
They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{189}$$

Convolutional Neural Network (CNN) Architectures

As discussed previously, LeNet 5, one of the first CNNs involved 5 ×5 convolutions with stride 1. Subsequently pooling layers were applied at stride 2. In summary, the architecture involves convolution-pool-convolution-pooling-fully connected-fully connected. In review, a single convolution layer results in N activation maps, where N is he number of slice/filters. Pooling is a method of compression, in which in a given neighborhood, we take only the maximum.

AlexNet. AlexNet, shown below, involves the following architecture.



In summary, the architecture for AlexNet is: Conv1 - MaxPool1 - Norm1 - Conv2 - MaxPool2 - Norm2 - Conv3 - Conv4 - Conv5 - MaxPool3 - FC6 - FC7 - FC8. For aan input of 227 × 227 × 3 involves 96 11 × 11 filters, resulting in an output of size 55 × 55 × 96. In general, the output size of a convolutional filter is determined from the following:

$$W' = \frac{W + 2P - F}{S} + 1 \tag{190}$$

where W is the input size, P is the padding,, F is the size of the filter (usually 3), and S is the stride. The first layer of AlexNet involves 96 11×11 convolution filters applied at stride 4. The second layer is a 3×3 pooling layer applied at stride 2. Important aspects of this network include: (i) first use of ReLU, (ii) normalization layers, (iii) heavy data augmentation, (iv) dropout, (v) batched gradient descent, (vi) momentum, (vii) L2 weight decay, and (viii) 7 CNN ensemble.

Trainable Parameters. In general, the number of layers in different layers can be determined as follows. For a convolutional layer, the number of trainable parameters takes the form:

$$\text{trainable parameters } = ((m \times n \times d) + 1) \times k$$

where m and n represent the height and width of the convolution filter, d is the depth of the layer input, either equivalent to the number of channels in the input or the number of filters in the previous later), and k is the number of filters. Note that the addition of 1 accounts for the bias. Similarly, for a fully connected layer, the number of trainable parameters is:

$$\text{trainable parameters } = c * (p + 1)$$

where c is the number of neurons in the current layer, while p is the number of neurons in the previous layer (or number of input parameters). Again, the addition of 1 accounts for the bias term. Note that pooling layers have no associated trainable parameters.

AlexNet - ImageNet Classification with Deep Convolutional Neural Networks

AlexNet (known as ImageNet LSVRC-2010) was used on the ImageNet database and achieved error rates of 37.5% and 17.0% with 60 million parameters and 650,0000 neurons. While not the first convolutional neural network (CNN) (e.g. LeNet), it was paradigm shifting due to GPU implementation (of note, multiple non-linearities maintained). The ImageNet database has roughly 22,000 categories with a total of 15 million labeled images. The network also introduced Rectified Linear Units (ReLU), which allowed for significantly faster training with the non-linearities maintained. AlexNET also included a local normalization scheme prior to application of the ReLU non-linearity

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{y=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta \tag{191}$$

Pooling was used to reduce the width and height of each filter domain. Inputs and outputs are described above. AlexNET also used data augmentation to produce an artificial data set (e.g. translations, shearing, reflections, random batches, etc.). In addition, a second form of data augmentation was used to alter the RGB channels by using principle component analysis (PCA) on the set of RBG pixel values followed by addition of magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1. Therefore, toe ach RGB image pixel $I_{xy} = [I^R_{xy}, I^G_{xy}, I^B_{xy}]^T$ we add the following quantity

$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T \tag{192}$$

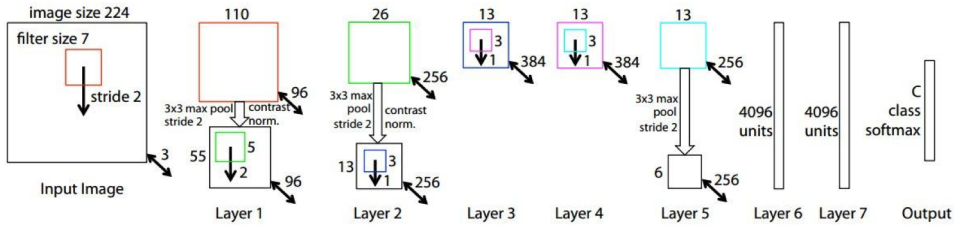
where $\mathbf{p}_i$ and λ_i are the ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values, respectively, and α_i is the aforementioned random variable. Each α_i is drawn only once for all the pixels of a particular training image until that image is used for training again, at which point it is re-drawn.

Additionally, standard momentum learning was used. Dropout was used to: (1) increase the size of the network, such that stochastically only a proportion of the neurons are learning (i.e. each neuron is forced to learn more) and (2) increase network generalizability. Combining the predictions of many different models is a very successful way to reduce test errors, but it appears to be too expensive for big neural networks that already take several days to train. Dropout is a very efficient version of model combination that only costs about a factor of two during training, consisting of setting to zero the output of each hidden neuron with probability 0.5. The neurons

which are "dropped out" in this way do not contribute to the forward pass and do not participate in back-propagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights. This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

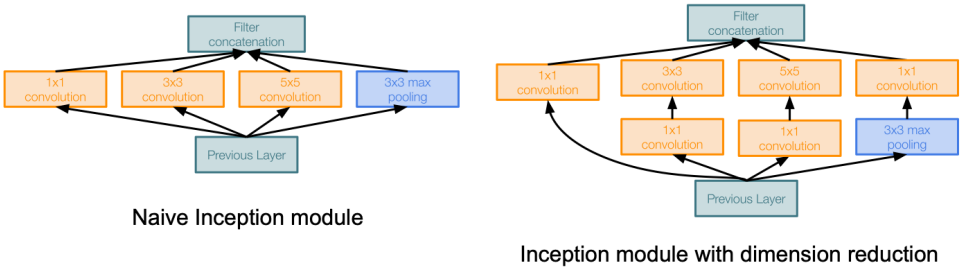
Other Neural Networks.

ZFNet, shown below, improved hyperparameters over AlexNet. Effectively, this network is AlexNet but changed the first layer to a 7×7 convolution with stride 2 and used more filters.



Next, we consider VGGNet, shown here. Smaller filters (stacks of 3×3 convolutions) have the same **effective receptive field** as one 7×7 convolution layer. However, networks are deeper with more non-linearities. They also have fewer parameters.

Another seminal network to consider was GoogLeNet, which allowed for deeper networks with computational efficiency. GoogLeNet had 22 layers with only 5 million parameters (fewer than AlexNet and VGGNet). It is an efficient inception module, with no FC layers. An inception module is a design involving a good local network topology (network within a network) and then stacks these modules on top of each other. GoogLeNet applies parallel filter operations on the input from previous layer: (a) multiple receptive field sizes for convolution, (b) pooling operation. Then all layers are concatenated together channel wise. Important to this network is the idea of a $1\times$ convolution, discussed in the next section. Briefly, this very expensive computationally. The pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer! As a result, GoogLeNet contains "bottleneck" layers that use 1x1 convolutions to reduce feature channel size. As shown, filter depth is reduced:



GoogLeNet therefore uses the same parallel layers as the naive example, and adding 1×1 conv, 64 filter bottlenecks, reducing the number of trainable parameters compared to 854M ops for naive version. Bottleneck can also reduce depth after pooling layer. The inception modules are stacked with dimension reduction on top of each other. Prior to the first inception module, a stem network (conv-pool-conv-conv-pool) is used. After the

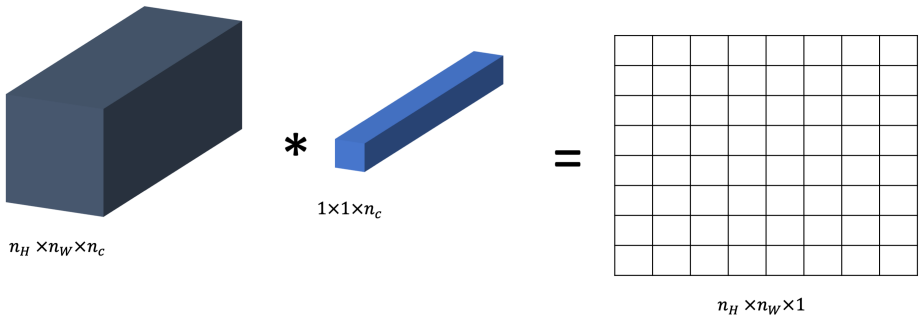
last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

One by One Convolution

Ostensibly a 1×1 convolution seems simply to be multiplying a number. Indeed in the 1D case,

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * [2] = \begin{bmatrix} 2 & 2 & 2 \\ 4 & 4 & 4 \\ 6 & 6 & 6 \end{bmatrix} \tag{193}$$

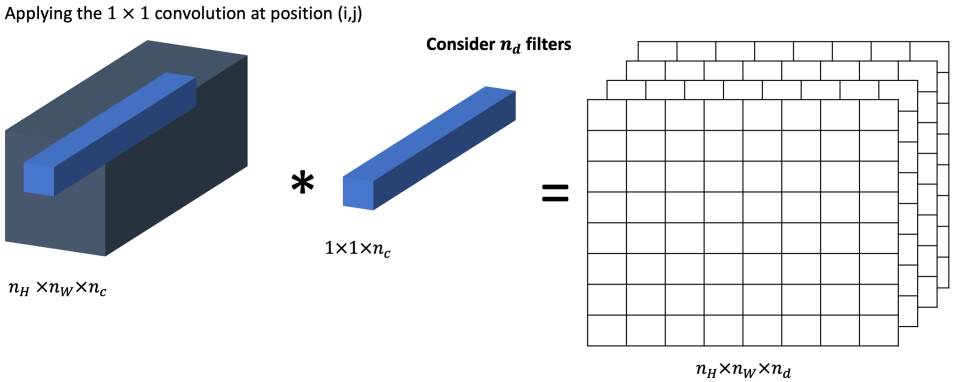
Its utility becomes recognized when looking at an $n_h \times n_w \times n_c$. Consider such an input. A 1×1 convolution here is denoted by the diagram below. Note that the 1×1 convolution has dimensions $1 \times 1 \times n_c$.



Note that here in the case of 1 filter, the 1×1 convolution compresses all the information for that single element. Following application of the 1×1 convolution, a non-linearity is applied. In general, consider a single 1×1 convolution filter with elements c_i. At the (i, j) position, the output of the convolution is the inner product between the elements in the convolution and the elements across all filters at the (i, j). Mathematically,

$$O(i, j, l) = \sum_{k=1}^{n_c} c_k^{(l)} I(i, j, k)$$

Here, n_c is the depth in the filter dimension, (i, j) is the row, column designation of the element, $c_k^{(l)}$ are the elements in the lth 1×1 convolution filter, I is the input, and O is the output. For the general case, a 1×1 convolution with n_d filters will result in an output with $n_H \times n_W \times n_C$. Shown below is a diagrammatic representation of the 1×1



Of note, 1×1 convolution is equivalent to a filly connected neural network layer across, with vectors across the filter dimension being the input. For this reason, 1×1 convolution also go by the name **network in network**.

We can do 1 of 3 things with a 1×1 convolution. We can either decrease $(n_c > n_d)$, maintain $(n_c = n_d)$, or increase $(n_c < n_d)$ the size of the filter dimension. Let us consider the first of these operations. A 1×1 convolution

with 32 filters will shrink the output from $n_H \times n_W \times n_c$ to $n_H \times n_W \times 32$. Akin to pooling, which shrinks the $n_h \times n_w$ dimension, 1×1 convolution shrinks the filter dimension. The primary utility here is that we are able to save on computation in cases when the filter dimension is too large. For the other two cases (either maintaining or expanding the filter dimension), the primarily utility here is that with a relatively simple operation with few parameters, we are introducing a nonlinearity. This allows the network to learn more complicated function.

1×1 convolutions have demonstrated utility in many networks, including InceptionNet and ResNet. First, consider InceptionNet. Each inception block in InceptionNet uses both 5×5 and 3×3 convolutions, running in parallel in the network. Especially for the 5×5, this is computationally quite expensive. Consider the following case within InceptionNet:

Imagine that we are going from an input with $28 \times 28 \times 192$ to an output of $28 \times 28 \times 32$ using 5×5 convolutions.

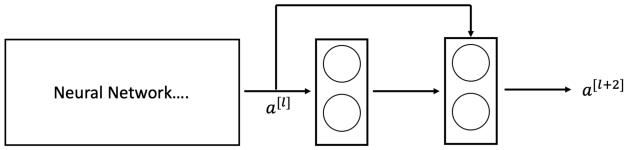
$$[28, 28, 192] \xrightarrow{5\times5} [28, 28, 32]$$

Here, we need approximately 120 million trainable parameters. Now consider adding a 1×1 convolution first, such that we have the same output size.

$$[28, 28, 192] \xrightarrow{1\times1} [28, 28, 16] \xrightarrow{5\times5} [28, 28, 32]$$

Here, we only need 12.4 million parameters. We are significantly saving on computation, while simultaneously introducing non-linearities. This is also known as a bottleneck layer. Thus, in InceptionNet, introduction of 1×1 convolutions allows for computational efficiency.

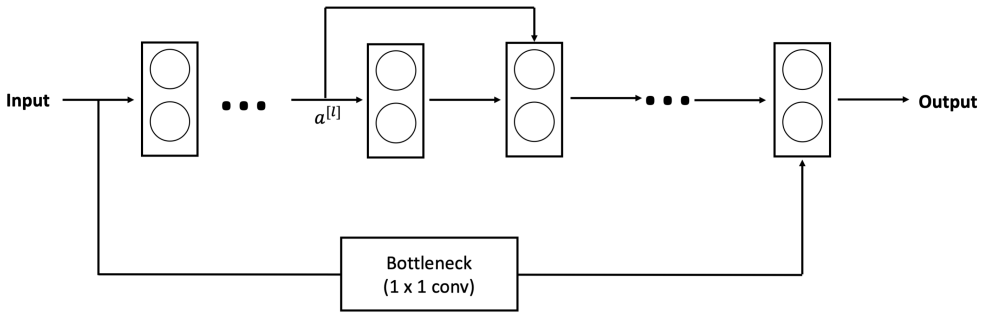
The first formulation of ResNet does not have a 1×1 convolution. However, subsequent iterations have incorporated a so-called bottleneck layer that passes the input to the output. In addition to introducing a non-linearity, the utility of a 1×1 convolution is that it allows for mapping from one layer to another, allowing for dimensional consistency. To better understand this, consider the following network layers, with assumed ReLu activation functions $g(\cdot)$:



Note that here

$$a^{[l+2]} = g(W^{[l+2]} a^{[l]} b^{[l+2]} + W_s a^{[l]})$$

where $W^{[l+2]}$ are the learned weights in the layer, $b^{[l+2]}$ are the corresponding bias terms, and W_s is some matrix that allows for dimensional consistency between the matrix $W^{[l+2]} a^{[l]} b^{[l+2]}$ and $a^{[l]}$ (i.e. so they can be added together. Now, consider a resnet, with a bottleneck, shown below:

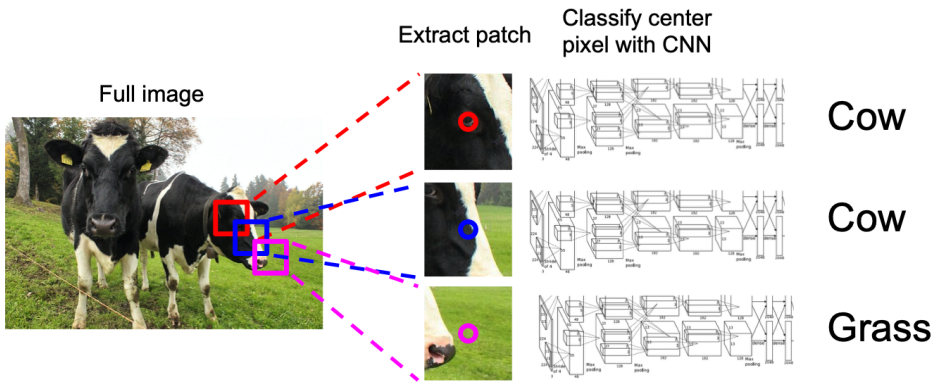


Here the 1×1 convolution has two functions: (a) It functions much like W_s in the example above, allowing for dimensional consistency between the input and the output, and (2) It allows for more efficient feature representation.

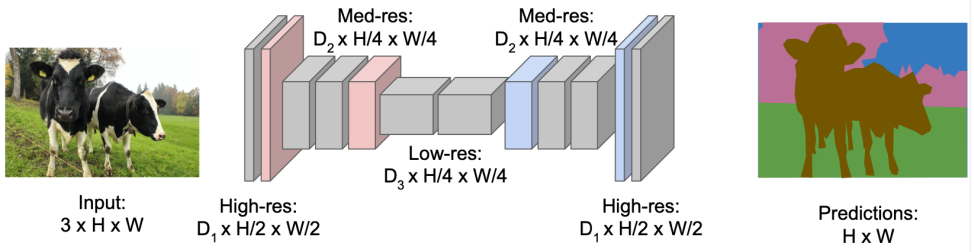
More on Resnet involves very deep networks using residual connections. a 152 layer module was used for ImageNet, resulting in the best performance of all models. In ResNet, rather than modeling $H(x)$, we are modeling $F(x) = H(x) - x$ and use an identity transformation. This transformation is known as a residual block, as this connection repeats throughout the network. What happens when we continue stacking deeper layers on a "plain" convolutional neural network? Deep models have more representation power (more parameters) than shallower models. However, the problem is an optimization problem, deeper models are harder to optimize. The solution is to use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping. This allows for easy learning of the identity mapping: $H(x) = x$ if $F(x) = 0$. ResNet uses layers to fit the residual $F(x) = H(x) - x$ instead of $H(x)$ directly. The full ResNet architecture includes: stacked residual blocks, with every residual block with two 3×3 convolution layers. Periodically, the number of filters is doubled and down-sampled spatially using stride 2 ($/2$ in each dimension) to reduce the activation volume by half. A stem layer at the beginning with additional layers is sometimes added. For deeper networks (ResNet-50+), use "bottleneck" layer (1×1 convolutions) to improve efficiency (similar to GoogLeNet). Batch normalization is also used. Good Practices for Deep Feature Fusion include Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models.

Detection and Segmentation

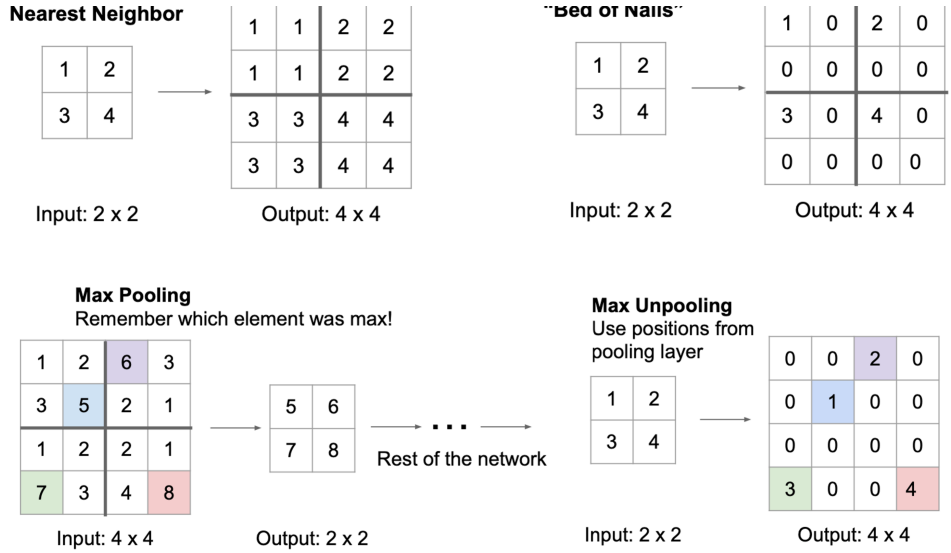
Computer vision tasks are varied and include classification, semantic segmentation, object detection, and instance segmentation. In general, semantic segmentation involves labeling each pixel in the image with a category label. We only care about pixels. Implementation involves a sliding window implementation, show below:



We therefore want to design a network as a bunch of convolutional layers to make predictions for pixels all at once! However, this is very expensive. As a result, we design network as a bunch of convolutional layers, with downsampling and upsampling inside the network, a so called hourglass architecture!

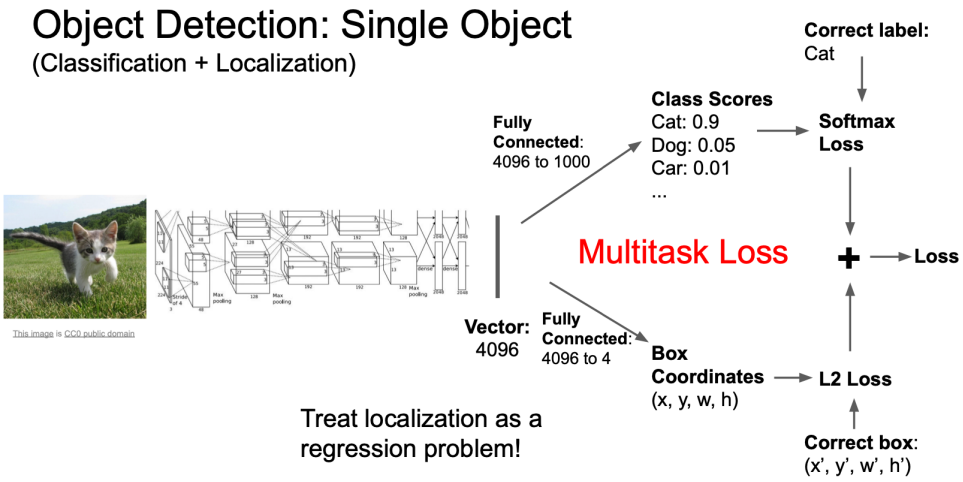


There are multiple methods for unpooling/upsampling including nearest neighbor, "bed of nails," max unpoooling and the transpose convolution, all shown below:

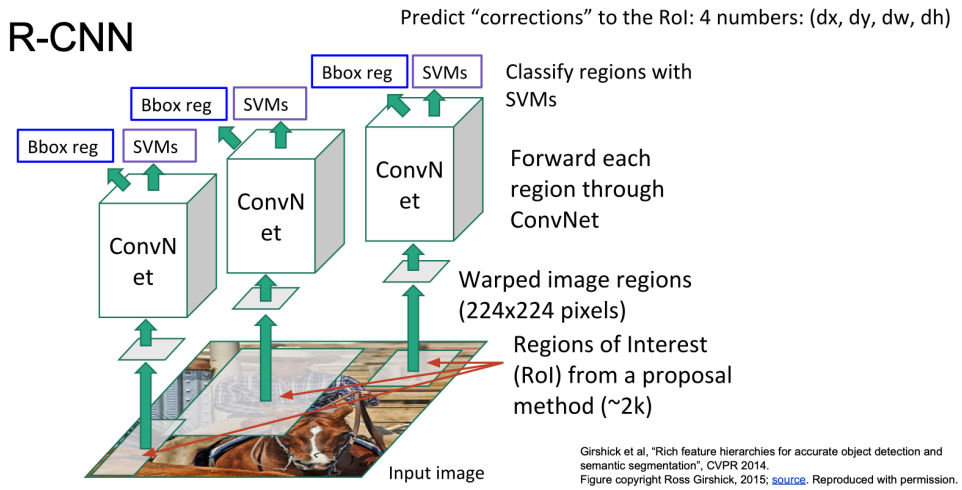


Object Detection

In brief, object detection involves classification and localization. We can therefore treat localization as a regression problem, as shown below;



Networks involved include R-CNN, fast R-CNNs, faster R-CNNs, YOLO, etc. For each feature, we effectively have a single loss function. In general, the approach involves application of a CNN to many different crops of the image, with the CNN classifying each crop as object or background For multiple objects, the problem becomes applying CNNs to a huge number of locations, scales, and aspect ratios, very computationally expensive! The solution involves so-called region proposals.selective search. We find "blobby" image regions that are likely to contain objects. This is relatively fast to run (e.g. Selective Search gives 2000 region proposal in a few seconds on CPU. R-CNNs can then applied through each warped region, such that each ROI from a proposal method is passed through a ConvNet, with region classification using SVMs.

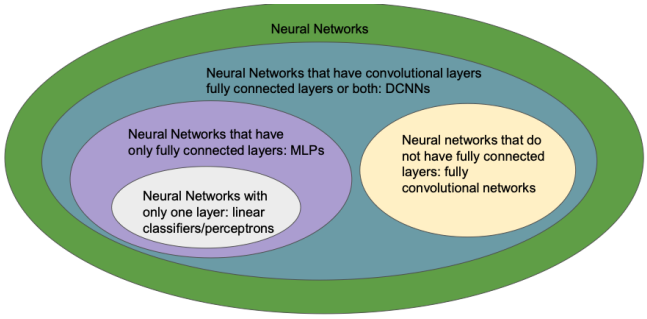


The problem here involves minimizing a detection cost and maximizing the region of overlap. The problem is that this is very slow. We need to approximately 2000 independent forward passes for each image. One solution would be to process the image before cropping (i.e. swapping the convolution and cropping process), known as Fast R-CNN.

Software and Hardware for DCNN Training

Recall that multilayer perceptions are units that operate using the dot product. The involve nonlinear activactions in each layer, e.g. ReLU, Sigmoid, tanh. Loss functions are used for training (e.g. perceptron loss, hinge loss (margin perceptron loss), quadratic loss). Deep convolutional neural networks (DCNNs) are multi-layer neural networks whose core operation is the convolution, with non linear activation, fully connected layers (MLP-style layers), and always trained with back propagation. Convolutions are used for a variety of reasons. Mixing convolutions and pooling is very good for describing images (recall Histogram Oriented Gradients/HOG). Convolutions are great for images. Dot product-based classification (perceptrons and MLPs) are fine, but they are for tabular data, not images. DCNNs learns now to represent the data (images). DCNNs are compositional ($f(g(x))$, much like MLPs. In MLPs, we have $a_i = \sigma(z_i)$ and $z_i = 2_1a_1 + ...w_na_n$. We leveraged the chain rule to compute the gradient of the loss with respect to all the parameters and used back propagation. The approach for DCNNs is the same. Convolutional operations to represent images (historical comparison) (e.g. HOG, SIFT, LBPs). Older DCNNs were functional (e.g. convolutions + pooling), but now the parameters are significatly more advanced. DCNNs are an idea from the late 80s. For historical context perceptrons are an idea from the late 50s/early 60s. DCNNs failed before because we did not have sufficient CPU/GPU computational power, ReLU, and big data. Regularizaiton methods, including early stopping, dropout, data augmentation, are also essential to the success of these systems. Regularization allows the solution to be either simplest (e.g. avoid overfitting) or find the hyperplane that best fits the data without over-fitting. In early stopping, we are minimizing the loss on the validation data. Dropout or batch gradient descent also improve robustness and therefore are regularization methods.

Training Starting Point: There are two orthogonal questions, where to start and what parameters to update: (a) from scratch: initialize the optimization from a random vector, and (b) from a pre-trained model: initialize the optimization from a vector that has already advanced in a related optimization procedure. We may either update all the parameters or updated only the parameters of the last layers. In general, the first few convolutional layers are performing the same



operations, so starting from a pre-trained model is always beneficial. Fine tuning/hyperparameter tuning is also necessary for convergence.

Multivariate Chain Rule: Recall that the multivariate chain rule reminder:

$$\frac{dy}{dx_i} = \sum_{s=1}^{N} \frac{\partial y}{\partial u_s} \frac{\partial u_s}{\partial x_i} \tag{194}$$

Using this fact, we next consider the gradient of a convolutional layer. Consider the following convolution:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \tag{195}$$

The numerical operation performed here is simply:

$$\begin{aligned} h_{11} &= w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22} \\ h_{12} &= w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23} \\ h_{21} &= w_{11}x_{21} + w_{12}x_{22} + w_{21}x_{31} + x_{22}x_{32} \\ h_{22} &= w_{11}x_{22} + w_{12}x_{23} + w_{21}x_{32} + x_{22}x_{33} \end{aligned} \tag{196}$$

The gradient of this convolutional layer is determined as follows. Let $L = \sum_{i,j}(t_{ij} - h_{ij})^2$ be the L2 loss. Note here the gradients are determined as follows:

$$\begin{aligned} \frac{\partial L}{\partial h_{11}} &= -2(t_{11} - h_{11}) & \frac{\partial L}{\partial h_{12}} &= -2(t_{12} - h_{12}) \\ \frac{\partial L}{\partial h_{21}} &= -2(t_{21} - h_{21}) & \frac{\partial L}{\partial h_{22}} &= -2(t_{22} - h_{22}) \end{aligned} \tag{197}$$

We also know the gradients of each of the parameters with respect to the weights:

$$\begin{aligned} \frac{\partial h_{11}}{\partial w_{11}} &= x_{11} & \frac{\partial h_{11}}{\partial w_{12}} &= x_{12} & \frac{\partial h_{11}}{\partial w_{21}} &= x_{21} & \frac{\partial h_{11}}{\partial w_{22}} &= x_{22} \\ \frac{\partial h_{12}}{\partial w_{11}} &= x_{12} & \frac{\partial h_{12}}{\partial w_{12}} &= x_{13} & \frac{\partial h_{12}}{\partial w_{21}} &= x_{22} & \frac{\partial h_{12}}{\partial w_{22}} &= x_{23} \\ \frac{\partial h_{21}}{\partial w_{11}} &= x_{21} & \frac{\partial h_{21}}{\partial w_{12}} &= x_{22} & \frac{\partial h_{21}}{\partial w_{21}} &= x_{31} & \frac{\partial h_{21}}{\partial w_{22}} &= x_{32} \\ \frac{\partial h_{22}}{\partial w_{11}} &= x_{22} & \frac{\partial h_{22}}{\partial w_{12}} &= x_{23} & \frac{\partial h_{22}}{\partial w_{21}} &= x_{32} & \frac{\partial h_{22}}{\partial w_{22}} &= x_{33} \end{aligned} \tag{198}$$

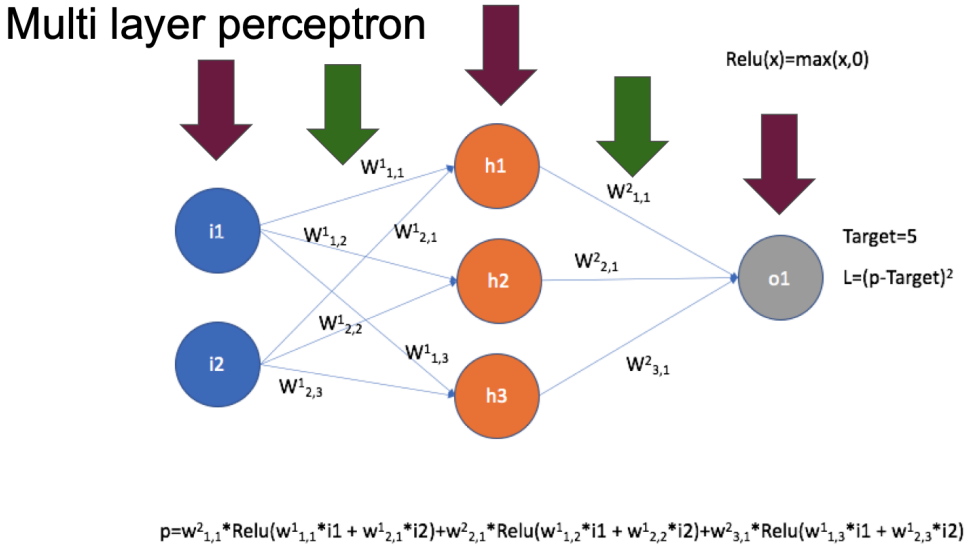
Using the chain rule, we can then substitute and write a final equation for each of the four partial derivatives (there are four weights).

$$\begin{aligned} \frac{\partial L}{\partial w_{11}} &= \textstyle\sum_{ij} \frac{\partial L}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial w_{11}} \\ &\vdots \\ \frac{\partial L}{\partial w_{22}} &= \textstyle\sum_{ij} \frac{\partial L}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial w_{22}} \end{aligned} \tag{199}$$

In general, the approach can be expanded to an activation function. Let $a_{ij} = g(h_{ij})$ be the ReLU activation with derivative g'. Then the gradients are simply

$$\begin{aligned} \frac{\partial L}{\partial w_{11}} &= \sum_{ij} \frac{\partial L}{\partial a_{ij}} \frac{\partial a_{ij}}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial w_{11}} \\ &\vdots \\ \frac{\partial L}{\partial w_{22}} &= \sum_{ij} \frac{\partial L}{\partial a_{ij}} \frac{\partial a_{ij}}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial w_{11}} \end{aligned} \tag{200}$$

where $\partial a_{ij}/\partial h_{ij}$ is simply g'. Recall that we have the same algorithmic approach for a multilayer perceptron.



Tools to drain DCNNs include Caffe, PyTorch, and Tensorflow. Fancy tools some peoples use include Mxnet and Fast.ai. At its core, PyTorch is a system to quickly compute the gradient of vector valued functions. It allows for quick prototyping, training, CUDA GPU support, flexibility, reasonable inference speed, python implementation, many layers, many networks, pretrained models available. In general, PyTorch is a good system all around.

Jacobian: Recall the Jacobian

$$J^T \cdot v = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \begin{pmatrix} \frac{\partial l}{\partial y_1} \\ \vdots \\ \frac{\partial l}{\partial y_m} \end{pmatrix} = \begin{pmatrix} \frac{\partial l}{\partial x_1} \\ \vdots \\ \frac{\partial l}{\partial x_m} \end{pmatrix} \tag{201}$$

Observe that we are talking about a few matrix multiplications at the core and a simple update rule for an algorithm, shown below. These could be implemented by hand, though not useful for system development. PyTorch provides tremendous value: Autograd for primitive functions, many layer types, fast implementation, and representation.

Gradients and Training: Computing the gradient is essentially training. The missing ingredient is called Stochastic Gradient Descent (SGD), which uses the gradient to minimize the loss; this procedure is commonly known as training. Consider a multivariate function F, which is differential of a neighborhood a. We start at a_0 and do the following:

$$a_{n+1} = a_n = \gamma \nabla F(a_n) \tag{202}$$

We can build a sequence: $F(a_0) >= F(a_1) >= F(a_2) >= ....$ If this converges, we found a local minima. If F is convex the local minimal will be a global minima. Now assume we have many data points and each of them has a loss, we can define the total loss as:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w) \tag{203}$$

We can then apply gradient descent on $Q(w)$, where w is the variable we are updating:

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n \tag{204}$$

At one extreme, we can compute the loss element by element (drop the sum), which is known as stochastic gradient descent:

$$w : -w - \eta \nabla Q_i(w) \tag{205}$$

Or, we can go batch by batch, where n is the batch size, which is smaller than the entire dataset, as in gradient descent. In practice SGD is not run with either the entire dataset, in which case it would not be stochastic, or one one example at a time. In practice batches are used, as this exploits GPU parallelism, enables faster convergence, and allows for situations where the dataset does not fit into the memory. Once the descent direction is computed, the learning rate determines how big a step to use: schedules start with a learning rate of 1, then after some iterations go to 0,1, then 0.01. Finding a good schedule is like finding a good batch size, i.e. it is critical for training success.

As we saw before, training and inference require fast matrix multiplication. The fastest way we have to do this is with a computational GPU, allowing for training with large volumes of data, fast, real time inference, and networks to be self contained and shared. Lines of GPUs are separated into Prosumer (Nvidia 1080, 2080, forthcoming 3080 and 3090, in the $1000-$2000 range, and Enterprise: in the $5000-$10000 range. In the past few years open exchange formats have become popular, these allow different training and inference engines to inter-operate: The most common one is ONNX (Open Neural Network Exchange). There are also tools that specialize in inference (not training) (NVidia TensorRt, Intel OpenVINO, and OpenCV has an inference tool for CPU).

Deep Representations for Face Analytics

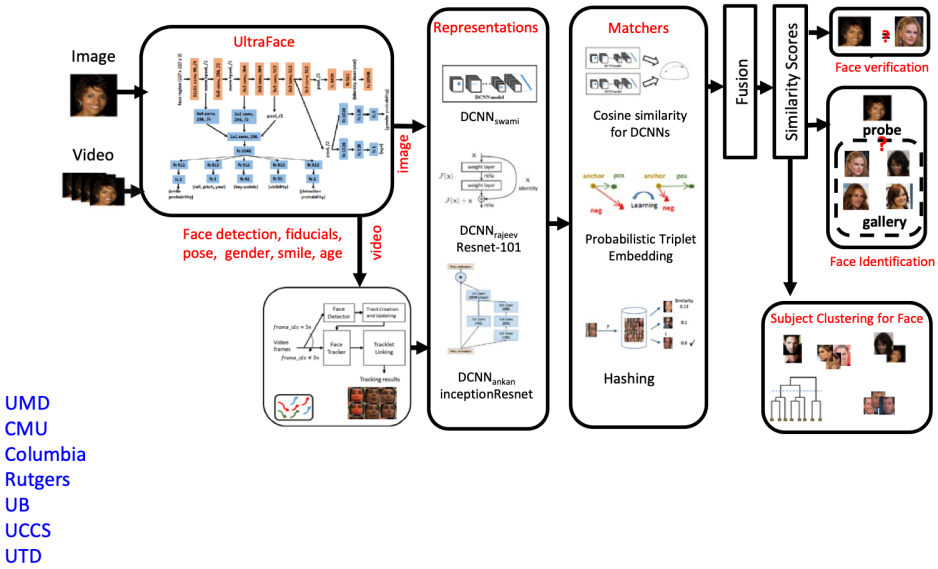
In addition to traditional computer vision methods, artificial neural networks were attempted for face analytics in the mid 80s-90s, representing early attempts at using deep networks. Since 2012, the emergence of large annotated data, GPUs, and approximation of nonlinearities have resulted in applications in deep learning pulling theory. Applications include unconstrained face verification/recognition, face analytics (Attributes, clusters, gender, expressions), object detection/recognition, and action detection/recognition. Neural networks have been around for a while and had a second life in the 80s with the introduction of Hopfield Networks. 1987 ICNN in San Diego introduced NNs for DARPA's ATR program. Examples of NNs in vision include Autonomous driving(Dean Pomerleau), Face detection(Tommy Poggio), and OCR (mostly in NIPS). Between 1996-2001, Grossberg introduced MURI on NNs for vision. By adding more layers, deep learning networks are beating SVMs (e.g. LeNet and AlexNett in 2012).

In general, the pillars of deep learning are:

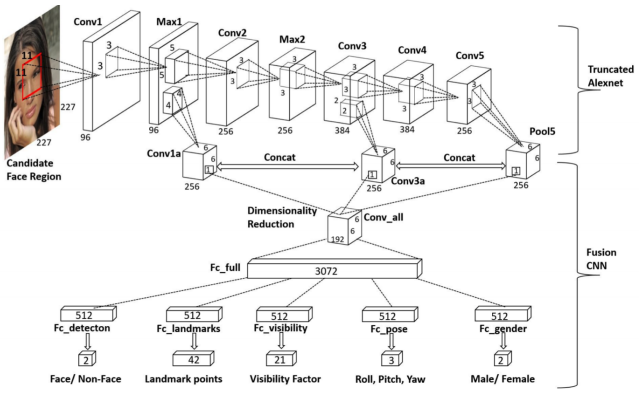
(1) Sufficient computational power: usually available.
(2) A suitable network architecture: how to design one?
(3) An abundance of training samples: not always the case.
(4) Perfect, noiseless training labels: human error is always present.
(5) Balanced training data across various classes: not always the case.
(6) Lossless training and testing samples: not always the case.
(7) Trusted and secure agent at deployment: there are real security threats!

Past attempts at face recognition included Fiducials, iso-depth curves, PCA (eigen pictures; eigen decomposition of the face), LDA (better at discriminating features), spherical harmonics and variants, neural networks, 3D morphable models (for faces that aren't looking at the camera), local binary patterns, spare representations and dictionaries. Since 2014, the problem has shifted towards unconstrained face verification and involved face verification, search, detection, subject specific clustering, known as JANUS. Very high levels of performance were

required. The original pipeline included attributed extraction, fiducial extraction, and feature learning, though not deep learning. However, with the rise of AlexNet, this entirely replaced with DCNN training/testing. The final pipeline, shown below utilized deep learning for representation, similarity scores, clustering, etc. There were two models, one based on InceptionNet and the other based on ResNet. After determination of feature, then a feature vector was compared, which was reduced to 128 and compared to a database.



Unique features of this effort included: multi-task learning in deep networks (face and gender detection, pose and age estimation, fiducial extraction, network of networks (fusion of short and tall networks, relatively small training data sets, a 384 float templates, sublinear search, easy training, state of the art performance on face verification, search, and clustering tasks, an extensive publication record, and implications in forensic science. The hyperface face architecture allowed for identification of face, landmarks, visibility, roll, pitch, yaw, and gender specification, attached to a truncated AlexNet.



Multi-task Learning regularizes the shared parameters of the network and reduces overfitting. This enables the network to learn the correlations between data from different distributions in an effective way. Alignment is also important, and a similarity transform was used (rotation, translation independent). Similar to hyperface, the ultraface architecture has parameters initialized from face identification network. Subject-independent tasks (face-detection, fiducials, pose, smile) share the lower layers of the network. Subject-specific features are pooled from deeper layers of the network. Ultraface regularization was either task based (shrink the solution space of θ_s such that the learned parameter vector is in consensus with all the tasks), or domain based (θ_s adapts to the complete set of domains instead of fitting a task-specific domain). Compared hyperface, ultraface had better initialization, domain based regularization, increased number of tasks, and improved results for all the tasks.

The training data set included CASIA, MORPH, IMDB, Adience, CelebA, and AFLW. Cross entropy loss was used along with euclidean loss weighted with visibility. For video based recognition, a deep pyramid single shot face detector (DPSSD) was developed, with an hourglass architecture. Face associations in videos involved face tracking, etc. Essential to training was to minimize the softmax loss function:

$$\text{argmin } -\frac{1}{M}\sum_{i=1}^{M}\log\frac{e^{W_{y_i}^T f(\mathbf{x}_i+b_{y_i})}}{\sum_{j=1}^{C} e^{W_j^T f(\mathbf{x}_i)+b_j}} \quad \text{subject to } ||f(x_i)||_2=\alpha,\ \forall\ i=1,2,...,M \tag{206}$$

where the softmax loss was constrained to a hyper-sphere, resulting in a feature normalization (i.e. L2 softmax). We can adopt a probabilistic interpretation as follows. If the underlying class density functions follow a Von Mises-Fisher Distribution

$$f_p(\mathbf{x};\boldsymbol{\mu},\kappa)=C_p(\kappa)\exp(\kappa\boldsymbol{\mu}^T\mathbf{x}) \tag{207}$$

$$C_p(\kappa)=\frac{\kappa^{p/2-1}}{(2\pi)^{p/2}I_{p/2-1}(\kappa)} \tag{208}$$

$$\kappa\geq 0, \quad ||\boldsymbol{\mu}||=1 \tag{209}$$

where p is the feature dimension and x is the normalized feature vector. On the hypersphere, the features are well separated, and the hypersphere itself is a well defined manifold. We can determined the parameter κ using a Bayesian approach by means of determining the maximum a posteriori (MAP) estimate:

$$\mathbf{L}=\text{maximize } \log\frac{f_p(\mathbf{x}_i;\boldsymbol{\mu}_i,\kappa)}{\sum_{j=1}^{C} f_p(\mathbf{x}_j;\boldsymbol{\mu}_j,\kappa)}=\text{maximize } -\log\frac{\exp(\kappa\boldsymbol{\mu}_i^T\mathbf{x}_i)}{\sum_{j=1}^{C}\exp(\kappa\boldsymbol{\mu}_j^T\mathbf{x}_j)} \tag{210}$$

The loss function is extremely important for convergence. The UMD database used as part of training included 378,839 face annotations of 8,501 subjects; Face box and identity annotations verified by humans; Wide range of poses and expressions; Annotations also provided for fiducial keypoints, 3-D pose, and gender generated by UltraFace. Multiple neural networks can be combined (mixture of experts model/conditional fusion); in most cases, this isn't much more efficacious as the dominant model dictates the outcomes.

Triplet Similarity Embedding: Define a triplet (a,p,n), where a, the anchor, and p, the positive are from the same class, but n, the negative belongs to a different class. The objective is to make (a,p) "more similar" and (a,n) "more dissimilar" in low dimensional space (like a linear discriminate analysis

$$(Wa)^T\cdot(Wp)>(Wa)^T\cdot(Wn) \tag{211}$$

We optimize the transformation W over the triplets from the training set

$$\underset{W}{\text{argmin}}\sum_{a,p,n\in\mathbb{T}}\max(0,\alpha+a^TW^TWn-a^TW^TWp) \tag{212}$$

where α is a margin parameter that defines a neighborhood around the anchor. The solution can be found using stochastic gradient decent. For good generalization is hard negative mining:

$$W_{t+1}=W_t-\eta*W_t*(a(n-p)^T+(n-p)a^T) \tag{213}$$

In hard mining, we are providing difficult examples in training, with east test cases.

In summary, tasks involved for testing face analytic algorithms were varied and included:

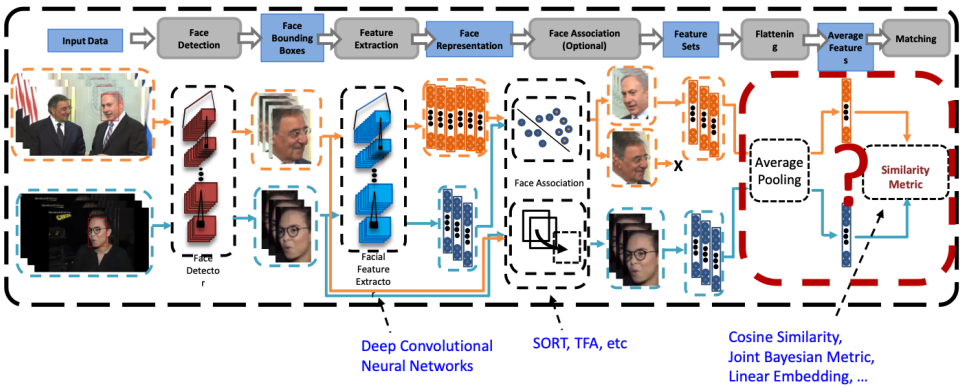
(1) Mixed Verification: This protocol consists of 15.79 million comparisons between mixed-media templates (images and video frames). It has about 3,548 unique subjects with a total of 23,221 templates. There are 19,673 genuine and 15,770,621 impostor matches.
(2) Covariate Verification: This protocol contains 47,795,011 pair of $1:1$ comparison using single image-based templates (7,846,524 genuine and 39,948,487 poster pairs)

(3) *N* Mixed Search: The gallery data is divided into two disjoint splits. Split 1 in the gallery contains 1,782 subjects and split 2 contains the rest of the 1,766 subjects. Probe data contain all 3,548 subjects with 459,621 probe templates.
(4) Face Detection: The protocol consists of 148,881 unique files (stills/frames/nonface images) with 262,305 faces.
(5) Subject Clustering: The clustering protocol is designed to test an algorithm's ability to identify multiple instances of the same subject from a collection of various pieces of media. In experiment 7, there are four clustering sub-protocols with 32 subjects (956 faces), 1,024 subjects (41,287 faces), 1,845 subjects (71,694 faces), and 3,548 (141,339 faces).
(6) Wild Probe with Still Image and Frames: The task is to identify subjects of interest from a collection of 9,693 still images, 117,543 frames, and 10,044 non-face images. It requires to detect faces and search each detected face against the CS4 galleries. Then, 1:N search test is evaluated between the templates and the curated CS4 galleries (G1 and G2).
(7) Wild Probe with Full Motion Video
(8) Wild Probe Mixed: The wild probe mixed protocol is designed to test an algorithm's ability to create and match templates from a collection of media (9693 still images, 10,044 non- face images, and 11,799 full-motion video) to a curated gallery. It requires to build templates for all identities determined to be within the collection of media by creating templates from single images or videos, clustering those templates, and then creating multi-media templates from the clustering output.

Data sets for testing include: IJB-A, CS3, CS4, etc. most of which are publicly available. Other data sets were more challenging and with more distractors (IJB-C, CSInfinity, MegaFace, etc.). Only deep learning algorithms can succeed on these data sets. Ensembles of deep learning networks further improved accuracy. Surprisingly compared with exert forensic examiners the algorithm was performing better.

Video Based Face Recognition

Prior to deep learning, algorithms for video based face recognition included gabor jets and particle filtering, online appearance models and particle filters, temporal models e.g. HMM, ARMA, 3D models, manifolds, linear subspaces (e.g. Discriminative canoncial correlation), affine subspaces (e.g. affine hull, convex hull), covariance matrices, and dictionaries. Deep learning for video based face recognition has significantly improved performance. An example algorithm is shown below:



SORT was a real-time online tracking algorithm which approximates the dynamics with linear Gaussian state space models and associates detection bounding boxes in every frame using Kalman Filters. Target face association retrieves a set of representative face images in a given video that are likely to have the same identity as the target face. Methods for analysis included flattening frame based deep features, image set based method for unconstrained video based face recognition, incorporation of temporal information, and face association using the face and the body. The IJB-B data set was used for training. The IJB-B dataset contains about 22,000 still images and 55, 000 video frames spread over 1, 845 subjects. Evaluation is done for 1:1 verification, and 1:N identification. The IJB-B verification protocol consists of 8,010,270 pairs between templates in the galleries (G1 and G2) and the probe templates. Out of these, 8 million are impostor pairs and the rest 10,270 are genuine comparisons.

An alternative approach is to leverage the set information. Here, we aggregate face features into a unified fixed-size representation for efficient face recognition in order to exploit the correlation information in the sets and generate unified representations. RNN based methods and subspace based methods are both appropriate, though RNN based methods are expensive and only applicable to sequential data. Performance on IBJ-S was good, though dropped for surveillance to surveillance identification protocols.

In summary, for video based face analytics, 3D models were not needed due to deep learning, though GAN based synthesis may be ok. Motion information in videos was not needed, flattening of frame based features works just fine. Deep networks also have the benefit of ease of software portability.

Two Dimensional Discrete Gaussian Markov Random Field Models for Image Processing

Here we begin a discussion on the usefulness of two-dimensional (2D) Gaussian Markov Random Field (GMRF) models for image processing, including Markovianity on a plane, statistical inference in GMRF models, and their applications. We effectively desire a model that explains the statistical characteristics of the image lattice. Let $y(s)$ be the neighboring pixels such that the linear weighted combination of $\{y(s+r), r \in N\}$. Here, N is the additive noise, known as the neighbor set, which does not contain $(0,0)$. Restrictions on N yield known representations, including causal models, in which N is defined as the subset of the set $\{(i,j) : i \leq 0, j \leq 0, (i,j) \neq (0,0)\}$. Causal models result in recursive image processing algorithms. They can also be unified to generate "unilateral" models by including more neighbors. The unilateral models result when N is a subset of the non symmetric half plane S^+ defined recursively. In 2D, we must generalize the observation $y(s)$ to non-causal and bidirectional models to account for neighbors in all directions.

Gaussian Markov Random Field (GMRF) models are a class of 2D noncausal models. Let $y(s), s \in \Omega = \{s = (i,j); 0 \leq i,j \leq M-1\}\}$ be the observations from the image. We assume a 2D GMRF model, which characterizes the statistical dependency among pixels, requiring that

$$p(y(s)|\text{all } y(r), r \neq s) = p(y(s)|\text{all } y(s+r), r \in N) \tag{214}$$

where N is the appropriate symmetric neighbor set. The simplest case would be including only the pixels north, south, east, and west $N = \{(0,1),(0,-1),(-1,0),(1,0)\}$. Incorporating more pixels would result in higher order 2D GMRF models. Since 2D GMRF are defined only for symmetric neighborhoods, we can equivalently define an asymmetric neighborhood set N_s, such that if $r \in N_s$ then $-r \notin N_s$ and $N = (r : r \in N_s) \cup (-r : r : N_s)$. We require estimation of the parameters of the GMRF as well as the optimal N. Before tackling this problem, we first provide a review of the relevant theory of 1D models.

Suppose we have a set of discrete Gaussian observations $\{r(t), t = 1,2,...,N\}$. Then $\{r(\cdot)\}$ is said to be a unilateral mth order Markov process if

$$p\{r(t)|\text{ all } r(j), j < t\} = p\{r(t)|\text{ all } r(t-j), 1 \leq j \leq m\} \tag{215}$$

The sequence $\{r(\cdot)\}$ obeying the Markovian property has the representation

$$r(t) = \sum_{j=1}^{m} \theta_j r(t-j) + \sqrt{\beta}\omega(t), \quad t = 1,2,...,N \tag{216}$$

with associated initial conditions $\{r(0), r(-1), ..., r(1-m)\}$. Here, $\{\omega(\cdot)\}$ is IID Gaussian noise with mean zero and variance unity. The $\{r(\cdot)\}$ is said to be a unilateral mth order Markov process if

$$E(r(t)|\text{ all } r(j), j < t) = E(r(t)|\text{ all } r(t-j), 1 \leq j \leq m) \tag{217}$$

$$\text{var}(r(t)|\text{ all } r(j), j < t) = \text{var}(r(t)|\text{ all } r(t-j), 1 \leq j \leq m) \tag{218}$$

For any arbitrary continuous of $\{\omega(\cdot)\}$ the sequence $\{r(\cdot)\}$ is a wide sense unilateral mth order Markov process. Markovianity only implies strict sense Markovianity if the random variables are Gaussian. The sequence $\{r(\cdot)\}$ is said to obey an mth order bilateral process if

$$p(r(t)|\text{ all } r(i), i \neq t) = p(r(t)|r(t-1)...r(t-m), r(t+1), ..., r(t+m)) \tag{219}$$

The sequence $\{r(\cdot)\}$ obeying this representation has the form

$$r(t) = \sum_{j=-m}^{m} \theta_j r(t-j) + e(t) \tag{220}$$

where

$$\theta_k = \theta_{-k} \tag{221}$$

and $e(\cdot)$ is a correlated Gaussian noise sequence with the conditional structure

$$p(e(s)|\text{ all } r(t), t \neq s) = p(e(s)|\text{ all } r(s+t), -m \leq t \leq m, t \neq 0) \tag{222}$$

This implies a weaker condition involving second order properties, namely that

$$E(e(s)|\text{ all } r(t), t \neq s) = 0 \tag{223}$$
$$\text{var}(e(s)|\text{ all } r(t), t \neq s) = \nu \tag{224}$$

which in turn implies that the correlated noise sequence $e(\cdot)$ has the following correlation structure:

$$E(e(t)e(s)) = \nu, \quad t = s \implies E(e(t)e(s)) = \begin{cases} -\theta_{t-s}\nu, & |t-s| \leq m \\ 0, & \text{otherwise} \end{cases} \tag{225}$$

The bilateral Markov Model is driven by a correlated noise sequence. As a result

$$E(r(s)|\text{ all } r(t), t \neq s) = \sum_{i=-m}^{m} \theta_i r(s-i) \tag{226}$$

If $\{r(\cdot)\}$ obeys a unilateral model of order m then it also obeys a bilateral model of order m. Furthermore, any bilateral model possesses an equivalent unilateral model when spectral factorization applies.

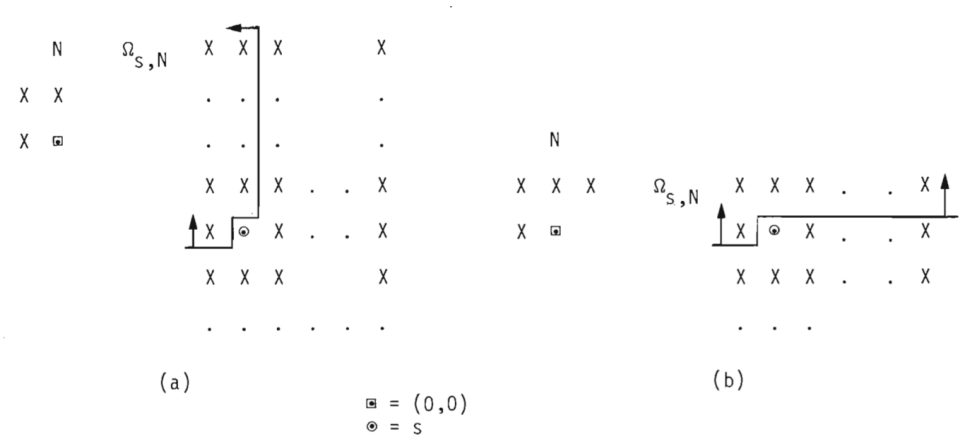
We next consider unilateral 2D GMRF models. Assume that the observations obeys the difference equation

$$y(s) = \sum_{r \in N} \theta_r y(s+r) + \sqrt{\beta}\omega(s) \;\forall\; s \tag{227}$$

Here, the set N is a associated with a subset of $\{(i,j) : i \leq 0, j \leq 0, (i,j) \neq (0,0)\}$ and the noise $\omega(s)$ is IID. Initial conditions are associated with the choice of N (e.g $\{(-1,0),(0,-1),(-1,-1)\}$ has initial conditions along the first row and column). Causal models are obtained by restricting N to a subset of the quarter plane. We generalize causal models as follows. Let S^+ be the nonsymmetric half plane, defined as: (1) $s \in S^+, r \in S^+ \implies r+s \in S^+$, (2) $s \in S^+ \implies -s \notin S^+$, and (3) $(0,0) \notin S^+$. Note that S^+ is not unique. With N being a subset of S^+, $\{y(s)\}$ is said to be strict sense unilateral Markov with respect to N if

$$p(y(s)|\text{ all } y(r), r \in \Omega_{s,N}) = p(y(s)|\text{ all } y(s+r), r \in N) \tag{228}$$

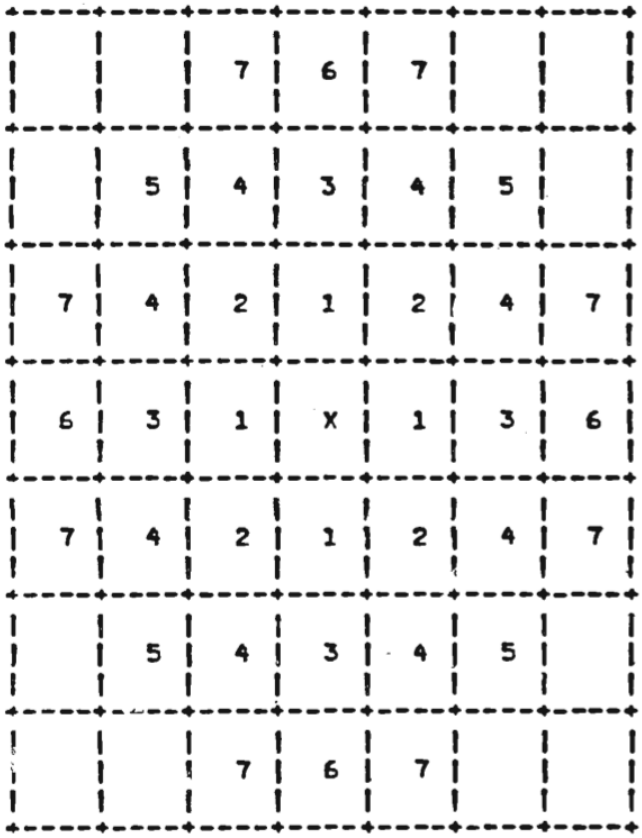
where $\Omega_{s,N}$ is defined such that: (1) $s \notin \Omega_{s,N}$, (2) $s+r \in \Omega_{s,N}$ for all $r \in N$, and (3) $r \in \Omega_{s,N} \implies (r+t) \in \Omega_{s,N}$ for all $t \in N$ provided $r+t \neq s$. Example structures of $\Omega_{s,N}$ are shown below:



Most image analysis requires non-cause or bilateral 2D GMRF models, as the pixel of interest may be dependent on any of the surround pixels (e.g. north, south, east, west, diagonals, etc.). Given an observation set $\{y(s)\}$ that obeys a non causal model whose structure does not factorize, we can only use approximate finite order representations of data. Let $\{y(s)\}$ be strictly Markov with respect to a symmetric neighborhood set N, then

$$p(y(s)|\text{ all } y(r), r \neq s) = p(y(s)|\text{ all } y(s+r), r \in N) \tag{229}$$

The figure below shows what different orders of such a GMRF would look like.



We can consider such an observation to be wide set Markov if the following relationships hold the expectation and variance with respect to N:

$$E(y(s)|\text{ all } y(r), r \neq s) = E(y(s)|\text{ all } y(s+r), r \in N) \tag{230}$$
$$\text{var}(y(s)|\text{ all } y(r), r \neq s) = \text{var}(y(s)|\text{ all } y(s+r), r \in N) = \nu > 0 \tag{231}$$

Let P be the minimum width of contiguous lattice points in Ω^+ and outside Ω^- such that all points in Ω^+ are P from every point in Ω^-. Let this region be $\partial\Omega$. The set $y(s)$, $s \in \Omega^+$ is a Markov order P if

$$p(y(s), s \in \Omega^+ | y(s), s \in \Omega^-, s \in \partial\Omega) = p(y(s), s \in \Omega^+ | y(s), s \in \partial\Omega) \tag{232}$$

There are two approaches to modeling this image: (1) We can either consider the image as a finite lattice that is a subset of an infinite lattice model, or (2) we can define a finite lattice model. We will consider each here.

Infinite Lattice GMRF Models: Assume that the observations $\{y(s), s \in \Omega\}$ have zero mean, Gaussian distribution that obeys

$$y(s) = \sum_{r \in N_s} \theta_r (y(s+r) + y(s-r)) + e(s) \tag{233}$$

where the noise is Gaussian with the covariance matrix

$$E(e(s)e(r)) = \begin{cases} -\theta_{s-r}\nu, & (s-r) \in N \\ \nu, & s = r \\ 0, & \text{otherwise} \end{cases} \tag{234}$$

The set N_s is the asymmetric neighbor set mentioned. Furthermore, by orthogonality $E(e(s)y(r)) = 0, \quad r \neq s$ or ν outside the neighborhood. As a result, the conditional expectation of our error variance is:

$$E(e(s)|\text{ all } y(r), r \neq s) = 0 \tag{235}$$

which implies that

$$p(y(s)|\text{ all } y(r), r \neq s) = p(y(s)|\text{ all } y(s+r), r \in N) \tag{236}$$

Equivalently, we can use the definition of a Markov process for this same result. Recall

$$E(y(s)|\text{ all } y(r), r \neq s) = \sum_{r \in N} \theta_r y(s+r) \tag{237}$$

Let the error be

$$e(s) = y(s) - \sum_{r \in N} \theta_r y(s+r) \tag{238}$$

As $y(r)$ is orthogonal to the error

$$E(e(r)y(s)) = \begin{cases} 0, & r \neq s \\ \nu, & \text{for some } \nu, r = s \end{cases} \tag{239}$$

Since $y(s)$ is Gaussian, then we know that the conditional expectation is 0 as we are assuming that the error is Gaussian conditional upon the data (i.e. $E(e(r)|y(s), r \neq s) = 0$). Since we are required to have the covariance matrix

$$E(e(s)e(r)) = \begin{cases} -\theta_r\nu, & r = s - t \\ 0, & \text{otherwise} \end{cases} \tag{240}$$

the GMRF model is defined only for symmetric neighbor sets.

Finite Lattice GMRF Models. Finite lattice models require specific assumptions about the boundary pixels, making them easier to analyze, though less precise compared to infinite lattice methods. Here, we assume

doubly periodic boundary conditions. Consider a Gaussian MRF model characterized by some neighborhood N. For the first order case, $N = \{(0,-1),(0,1),(-1,0),(1,0)\}$, and $N_S = \{(0,1),(1,0)\}$. Let the lattice region Ω specify data samples in the finite lattice points. Assuming doubly-periodic boundary conditions, then the data $y(s)$ is:

$$y(s) = \sum_{r \in N} \theta_r (s \mod r) + e(s) \tag{241}$$

where $(s \mod r) = ([(s_1 + r_1) \mod M, (s_2 + r_2) \mod M])$, known as a toroidal lattice. $\{e(s)\}$ is a correlated noise sequence with the correlation. Its second moment is:

$$E(e(s)e(r)) = \begin{cases} -\theta_{r-s}\nu & \text{if } (s-r) \mod M \in N \\ \nu & \text{if } s = r \mod M \\ 0 & \text{otherwise} \end{cases} \tag{242}$$

Here, $\nu > 0$ is the variance of $e(s)$. We can represent our model $y(s)$ in vector-matrix notation using a symmetric block-circulation matrix $B(\boldsymbol{\theta})$:

$$B(\boldsymbol{\theta})\mathbf{y} = \mathbf{e} \tag{243}$$

where $\mathbf{y} = [y(0,0),...,y(0,M-1),y(1,0),...,y(1,M-1),...,y(M-1,M-1))]^T$ and $\mathbf{e} = [e(0,0),...,e(0,M-1),e(1,0),...,e(1,M-1),...,e(M-1,M-1))]^T$, both $M^2 \times 1$ vectors. Note that here as $B(\boldsymbol{\theta})$ is a symmetric block circulant matrix

$$B(\boldsymbol{\theta}) = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{12} \\ B_{12} & B_{11} & \cdots & B_{13} \\ \vdots & \ddots & \ddots & \vdots \\ B_{12} & B_{13} & \cdots & B_{11} \end{bmatrix} \tag{244}$$

with each $B_{i,j}$ itself being an $M \times M$ symmetric circulant matrix whose (m,n)th element $b_{i,j}(m,n)$ is:

$$b_{i,j}(m,n) = \begin{cases} -\theta_r & \text{if } r = (|m-n| \mod M, |i-j| \mod M) \in N \\ 1 & \text{if } i = j \mod M \text{ and } m = n \mod M \\ 0 & \text{otherwise} \end{cases} \tag{245}$$

Given this matrix definition, we can rewrite the second moment of the noise (equivalent to the covariance matrix as we noise has 0 mean) as

$$E\{\mathbf{e}\mathbf{e}^T\} = \nu B(\boldsymbol{\theta}) \tag{246}$$

By orthogonolity, we have

$$E\{\mathbf{y}\mathbf{e}^T\} = \nu I \tag{247}$$

$$E\{\mathbf{y}\mathbf{y}^T\} = \nu [B(\boldsymbol{\theta})]^{-1} \tag{248}$$

In order for this problem to be stable, the covariance matrix must be positive definite. As a result, we know that the eigenvalues μ_k for $(k_1,k_2) \in K = \{k; 0 \le k_1, k_2 \le M-1\}$ are also greater than 0. For a block circulant matrix, it can be shown that the eigenvalues are simply the discrete Fourier of the first row. As a result, we have the following stability condition for $\{y(s)\}$.

$$\mu_k = 1 - \boldsymbol{\theta}^T \boldsymbol{\phi}_k > 0, \quad k \in K \tag{249}$$

$$\boldsymbol{\phi}_k = \left[\exp\left(\frac{j2\pi k^T r}{M}\right); r \in N \right]^T \tag{250}$$

In summary the corresponding eigenvalues and eigenvectors, which are the corresponding Fourier nodes:

$$\begin{aligned}\mu_k &= 1 - \boldsymbol{\theta}^T \boldsymbol{\phi}_k \\ v_k &= \frac{1}{\sqrt{M}} \boldsymbol{\phi}_k \\ \boldsymbol{\phi}_k &= \left[\exp\left(\frac{j2\pi k^T r}{M}\right); r \in N \right]^T \end{aligned} \tag{251}$$

In summary, doubly periodic boundary conditions result in a GMRF model that is represented on a toroidal lattice, which is very useful given that the an infinite lattice GMRF can be constructed from finite toroidal lattice GMRFs.

Spatial Autoregressive (SAR) Models: An SAR model assumes the following representation of $y(s)$:

$$y(s) = \sum_{r \in N} \theta_y y(s+r) + \sqrt{\beta}\omega(s) \tag{252}$$

where $\beta, \theta_r, r \in N$ are unknown parameters and $\omega(\cdot)$ is IID noise with zero mean and unit variance. The neighborhood may not contain the origin. We also symmetry ($\theta_r = \theta_{-r}$) for $r, -r \in N$. The main difference is that the SAR mode is not Markov with respect to the noncausal neighbor set N. If the observations $\{y(s)\}$ are Gaussian, then we can construct a neighborhood for which the Gaussian SAR model is non causal Markov, though the converse is not always true.

Synthesis of Images Using GMRF Models: We assume the following representation $B(\boldsymbol{\theta})\mathbf{y} = \mathbf{e}$. As $B(\boldsymbol{\theta})$ is symmetric $M^2 \times M^2$, where M is the size of the image, then

$$\mathbf{y} = B^{-1}(\boldsymbol{\theta})\mathbf{e} \tag{253}$$

In the block circulant, then we can construct the inverse from its eigenvalues (DFT of the first row of the matrix). Recall that for $B(\boldsymbol{\theta})$,

$$\begin{aligned} f_k &= 1 - \boldsymbol{\theta}^T \boldsymbol{\phi}_k \\ t_k &= \frac{1}{\sqrt{M}} \boldsymbol{\phi}_k \\ \boldsymbol{\phi}_k &= \left[\exp\left(\frac{j2\pi k^T r}{M}\right); r \in N \right]^T \end{aligned} \tag{254}$$

Thus

$$\mathbf{y} = \frac{1}{M^2} \sum_{s \in \Omega} \frac{\mathbf{f}_s x_s}{\mu_s}, \quad \text{where } x_s = \mathbf{f}_s^T \mathbf{e} \tag{255}$$

Here, $\mathbf{e}$ is Gaussian with correlation matrix $B(\boldsymbol{\theta})$ and μ_s is its variance. We generate $\mathbf{e}$ as follows. Let $\omega(s), s \in \Omega$ be an array of zero mean unit variance white noise Gaussian arrays such that

$$\epsilon(s) = \omega(s)(\mu_s)^{1/2}, \quad \xi(s) = \frac{1}{M} \sum_{r \in \Omega} \exp(j2\pi M s^T r / M) \tag{256}$$

The correlation matrix of $\xi(s)$ is identical to that of $e(s)$. Then the we know that the inverse of this matrix is well known, as we know the eigenvalues of $B(\theta)$.

Estimation in GMRF Models. In general, for a noncausal GMRF model, it is true that

$$p(y(s), s \in \Omega) \neq \prod_{s \in \Omega} p(y(s) | \text{ all } y(s+r), r \in N) \tag{257}$$

As a result, the expectation of $y(s)$ conditioned on $y(r)$, where $r \neq s$ is a function of $y(s+r), r \in N$. The likelihood $p(y(s), s \in \Omega | \boldsymbol{\theta}\nu)$ is difficult to determine in most cases other than the Gaussian case. For the finite

lattice case with doubly periodic boundary conditions, this is simply because the Jacobian of the semicirculant matrix B is easily evaluated. The coding method avoids determining the likelihood. Consider a simple first order isotropic Gaussian Markov Random Field Model with $N = \{(0,-1),(0,1),(-1,0),(1,0)\}$.

Coding Method: Recall that for a first order GMRF model, its observations can take the form

$$y(s) = \sum_{r\ in N_s} \theta_r[y(s+r) + y(s-r)] + e(s),\ \forall\ s \tag{258}$$

Here, $e(s)$ is the zero mean, variance ν correlated noise sequence, where

$$E[e(s)e(r)] = \begin{cases} -\theta_{s-r}\nu & (s-r) \in N \\ \nu, & s = r \\ 0, & \text{otherwise} \end{cases} \tag{259}$$

Based on our neighborhood N, $N_S = \{(0,1),(1,0)\}$. Let $\boldsymbol{\theta}_r = [\theta_r, r \in N_S]^T$ be our parameter vector. The conditional distribution $p(y(s)|y(s+r'), r' \in N)$ takes the form of a Gaussian conditioned upon all data in the neighborhood

$$p(y(s)|y(s+r'), r' \in N) = \frac{1}{\sqrt{2\pi\nu}} \exp\Big[-\frac{1}{2\nu}\Big(y(s) - \sum_{r\in N} \theta_r y(s+r)\Big)^2\Big] \tag{260}$$

For the coding method, we estimate our parameter vector $\boldsymbol{\theta}$ as follows:

$$\boldsymbol{\theta}'_c = \Big[\sum_{\Omega_0} \mathbf{g}(s)\mathbf{g}^T(s)\Big]^{-1}\Big[\sum_{\Omega_0} \mathbf{g}(s)y(s)\Big] \tag{261}$$

$$\mathbf{g}(s) = \text{col}[g(s+r') + y(s-r'), r' \in N_S] \tag{262}$$

where Ω_0 is a subset of Ω consisting of alternating pixel within the field. Based on what is described above, the coding method yields an estimate of $\boldsymbol{\theta}'_{c'}$, $\boldsymbol{\theta}''_{c'}$ summed over $\Omega - \Omega_0$. Advantages of this method include that the likelihood need not be determined. However, a major disadvantage is that the estimates obtained are not efficient.

Least Squares Estimation. We will use a similar approach as above in determining the least squares estimate. However, instead of using Ω_0, we will choose to use a subset of interior pixels in Ω, Ω_I. Here, our estimate $\boldsymbol{\theta}^*$ is

$$\boldsymbol{\theta}^* = \Big[\sum_{\Omega_I} \mathbf{g}(s)\mathbf{g}^T(s)\Big]^{-1}\Big[\sum_{\Omega_I} \mathbf{g}(s)y(s)\Big] \tag{263}$$

$$\mathbf{g}(s) = \text{col}[g(s+r') + y(s-r'), r' \in N_S] \tag{264}$$

$$\nu^* = \frac{1}{M^2} - \sum_{\Omega_I} \big(g(s) - \boldsymbol{\theta}^{*T}\mathbf{g}(s)\big)^2 \tag{265}$$

The estimator $\boldsymbol{\theta}^*$ is an improvement to those determined from the coding method based on the following theorem. Note that this estimator is only asymptotically unbiased; therefore, consistency needs to be proved.

Theorem: Let $y(s)$, $s \in \Omega$ be observations obeying the GMRF model

$$y(s) = \sum_{r\ in N_s} \theta_r[y(s+r) + y(s-r)] + e(s) \tag{266}$$

Then, the following are true:

(1) The estimate $\boldsymbol{\theta}^*$ is asymptotically consistent.

(2) The asymptotic covariance matrix $\boldsymbol{\theta}^*$ is:

$$E[(\boldsymbol{\theta}-\boldsymbol{\theta}^*)(\boldsymbol{\theta}-\boldsymbol{\theta}^*)^T] = \frac{1}{M^2}\left[\nu\mathbf{Q}^{-1} + 2\nu^2(\mathbf{Q}^T\mathbf{Q})^{-1} - \frac{\nu}{M^2}\mathbf{Q}^{-1}\sum_s\sum_r \theta_{(s-r)}\mathbf{W}_{rs}(\mathbf{Q}^T)^{-1}\right] \tag{267}$$

$$\mathbf{Q} = E[\mathbf{g}(s)\mathbf{g}(s)^T] \tag{268}$$

$$\mathbf{W}_{r,s} = E[\mathbf{g}(r)\mathbf{g}(s)^T] \tag{269}$$

For the case of an isotropic first order Gaussian model with $N_s = \{(0,1),(1,0)\}$, the mean square error simplifies to

$$E[(\theta-\theta^*)^2] = \frac{2\theta^2(1-4\theta\rho_{10})}{4M^R\rho_{10}^2}, \quad \rho_{10} = \frac{\text{cov}(y(s), y(s+(1,0)))}{\text{cov}(y^2(s))} \tag{270}$$

As discussed, the estimator from the least square estimator is an improvement from that from the coding method, based on its covariance matrix.

Maximum Likelihood Estimation The ML estimator is determined by maximizing the likelihood distribution

$$\hat{\boldsymbol{\theta}} = \underset{\theta}{\text{argmax}} -\frac{2}{M^2}\log p(\mathbf{y}|\boldsymbol{\theta},\nu) = \frac{1}{(2\pi)^2}\int[\log S(\lambda,\boldsymbol{\theta})d\lambda] + \frac{1}{\mu}\left[C_0 - 2\sum_{r\in N_S}\theta_r C_r\right] + \log 2\pi \tag{271}$$

$$S(\lambda,\boldsymbol{\theta}) = \nu\left(1 - 2\sum_{r\in N_S}\theta_r\cos\lambda^T\cdot r\right)^{-1} \tag{272}$$

$$C_r = \frac{1}{M^2}\sum_{s\in\Omega} y(s)y(s+r) \tag{273}$$

In order for the model to the stable, the ML estimators must also satisfy the following conditions:

$$\left(1 - 2\sum_{r\in N_S}\theta_r\cos\lambda^T\cdot r\right) > 0 \tag{274}$$

$$\frac{1}{4\pi^2}\int\cos(\lambda^T\cdot r)S(\lambda,\hat{\boldsymbol{\theta}})d\lambda = C_{r'} \tag{275}$$

Despite its accuracy, the ML estimator is computationally difficult to calculate and converges for first and second order GMRF models after between 30 and 40 iterations. However, computational load can be reduced by reduced in the case of doubly periodic finite toroidal lattice boundary conditions.

Estimation of Parameters in Finite Lattice Models. Recall that for a finite lattice model, we assume a toroidal estimation for the data. The log likelihood for this representation is:

$$\log p(\mathbf{y}|\boldsymbol{\theta},\nu) = \sum_{s\in\Omega}\log(1-2\boldsymbol{\theta}^T\boldsymbol{\phi}_s) - (M^2/2)\log 2\pi\nu - \frac{1}{2\nu}\boldsymbol{y}^T B(\boldsymbol{\theta})\mathbf{y} \tag{276}$$

Alternatively, we can develop likelihoods for the GMRF for other cases as follows. We partition Ω into two mutually exclusive sets Ω_I and Ω_B, the boundary set:

$$\Omega_B = \{s=(i,j); s\in\Omega \quad \text{and } (s_r)\notin\Omega, r\in N\} \tag{277}$$

$$\Omega_I = \Omega - \Omega_B \tag{278}$$

For a first order Gaussian MRF, we now have two representations for our signal:

$$y(s) = \sum_{i=1}^{2} \theta_i (y(s+s_i) + y(s+\bar{s}_i)) + e(s), s \in \Omega_I \tag{279}$$

$$y(s) = \sum_{i=1}^{2} \theta_i (y_1(s+s_i) + y_1(s+\bar{s}_i)) + e(s), s \in \Omega_B \tag{280}$$

where

$$y_1(s+s_i) + y_1(s+\bar{s}_i) = \begin{cases} 2y(s+s_i) & \text{if } (s+s_i) \in \Omega, (s+\bar{s}_i) \notin \Omega \\ 2(s+\bar{s}_i) & \text{if } (s+s_i) \notin \Omega, (s+\bar{s}_i) \in \Omega \\ y_1(s+s_i) + y_1(s+\bar{s}_i)) & \text{otherwise} \end{cases} \tag{281}$$

As before, we can write the signal in the form $B(\boldsymbol{\theta})\mathbf{y} = \mathbf{e}$ with eigenvectors and eigenvalues:

$$\mu_{ij} = 1 + 2\theta_1\phi_1 + 2\theta_2\phi_2, \quad \phi_j = \cos(j\pi/M - 1) \tag{282}$$

$$\eta_{ij} = [\eta_i \phi_j \eta^j]^T \tag{283}$$

The log likelihood of this finite model is:

$$\log p(\mathbf{y}|\boldsymbol{\theta}, \nu) = \frac{1}{2} \sum_{i,j \in \Omega} \log \mu_{ij} - (M^2/2) \log 2\pi\nu - \frac{1}{2\nu} \sum_{i,j \in \Omega} ||z(\lambda_{ij})^2|| \mu_{ij} \tag{284}$$

Here $z(\lambda_{ij})$ are the 2D discrete cosine transforms of $\{y(s)\}$, whose modes are make the eigenvectors of $B(\boldsymbol{\theta})$.

Comparison of Estimators: Of all estimators, despite the relative ease of implementation of the Coding method, the least squares estimator is actually more efficient than both the Coding estimator and the ML estimator, though the ML estimator is the most accurate.

Image Representation

Image representation is extremely relevant given big data. The question becomes how to represent the image with a parametric model. In speech, data is represented as a time series auto regression model, so one approach would be to treat each row as a one dimensional signal; however, this is not satisfactory. We therefore require 2D models. There is some kind of causality when scanning models, extending estimation methods. This eventually gave rise to non symmetric half plane models, etc. However, note that 2D non causal models are more appropriate for image cases, as there is no real causality.

Here we will only consider stochastic models. Consider the 1D time series model $f(i), i = 1, ..., M$. We can write $f(i)$ as:

$$f(i) = \sum_{j=1}^{m} \theta_j f(i-j) + w(i) \tag{285}$$

where m is the model order. This is known as autoregressive model as it regresses on itself. We assume $w(i)$ has zero mean and is an independent noise variable. Note that these are all discrete observations. For this model

$$p(f(i)| \text{ all } f(j), j < i) = p(f(i)|f(i-1), ..., f(i-m)) \tag{286}$$

The signal only depends on the m previous components. We can get the θ_j's for this model using parameter estimation. Note that we can write our model as follows

$$f(i) = \boldsymbol{\theta}^t \mathbf{z}(i-1) + w(i) \tag{287}$$

$$\mathbf{z}(i-1) = [f(i-1)...f(i-m)]^T \tag{288}$$

One method for parameter estimation is the linear least squares estimator. Here we are attempting to minimize the following cost:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \frac{1}{N} \sum_{i=m+1}^{N} (f(i-1) - \boldsymbol{\theta}^T \mathbf{z}(i-1))^2 \tag{289}$$

where our initial conditions are $f(i), ..., f(m-1)$. Once we know $\boldsymbol{\theta}$, then we can recursively determine the signal given the noise. This is a generative model. The LLS estimator:

$$\hat{\boldsymbol{\theta}}_{LS} = \Big(\sum_{i=m+1}^{N} \mathbf{z}(i-1)\mathbf{z}^T(i-1) \Big)^{-1} \Big(\sum_{i=m+1}^{N} \mathbf{z}(i-1) f(i) \Big) \tag{290}$$

If the variance of $w(i)$ is ρ, then the LS estimator for the variance is:

$$\hat{\rho}_{LS} = \frac{1}{N-m} \sum_{i=m+1}^{N} \big(f(i) - \hat{\theta}_{LS}^T \mathbf{z}(i-1) \big)^2 \tag{291}$$

To determine the ML estimator, we assume that $w(i)$ is Gaussian such that $w(i) \sim \mathcal{N}(0, \rho)$. In this case, the likelihood distribution condenses to:

$$p(f(i), ..., f(N)|\boldsymbol{\theta}, \rho) = \frac{1}{(2\pi\rho)^{N/2}} \exp \Big\{ \sum_{i=m+1}^{N} (f(i) - \boldsymbol{\theta}^T \boldsymbol{z}(i-1))^2 \Big\} \tag{292}$$

We maximize the likelihood distribution with respect to our parameters to yield the estimates $\hat{\boldsymbol{\theta}}, \hat{\rho}$. Note that these are equivalent to the LS estimators for the Gaussian case. The residuals for this estimator are:

$$\hat{w}(i) = f(i) - \hat{\boldsymbol{\theta}}_{LS}^T \mathbf{z}(i-1) \tag{293}$$

For image and speech compression, we require the residuals. We can determine the residuals for all non-observed data. $\hat{w}(m+1), ..., \hat{w}(m)$. This is not true image compression. Image compression involves that we quantize $\hat{w}$ and substitute into the model, thought this approach is not as good as discrete cosine transform. Such an approach is similar to the methodology of generative adversarial networks (GANs) except it is a multilevel hierarchical non linear model with learned parameters.

Recall that we can determine the model order m using the Bayes Information Criteria (BIC):

$$BIC(k) = N \ln \hat{\rho}_k + m_k \ln N \tag{294}$$

where $\hat{\rho}_k$ is the ML estimator for the kth model and m_k is the number of parameters in the kth model. We choose k corresponding to the smallest BIC.

1D Non Causal Model

The non-causality idea will now be expanded upon. Consider a 1D non causal model, shown below:

$$f(i) = \sum_{k=-m}^{m} f(i-k) + e(i) \tag{295}$$

Here i depends on the past on the future. This is known as a bilateral or interpolated model. We must have symmetry and specify that $\theta_k = \theta_{-k}$. We must also have the following correlation structure for the noise process:

$$E(e(i)e(j)) = \begin{cases} \nu, & i = j \\ -\theta_{i-j}\nu, & |i-j| \leq m \end{cases} \tag{296}$$

$e(i)$ is a correlated noise process. It must be correlated in order for the bilateral Markov property:

$$p(f(i)| \text{ all } f(j), i \neq j) = p(f(i)|f(i-1)...f(i-m), f(i+1), ..., f(i_m)) \tag{297}$$

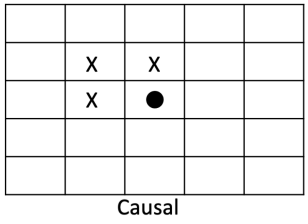
In one dimension, the non causal approach does not contribute to model accuracy, as we can find an equal causal model, though this is not the case in 2D. In fact, in 2D, non causal models are preferred.

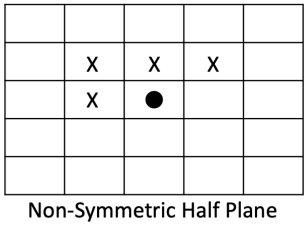
2D Non Causal Models

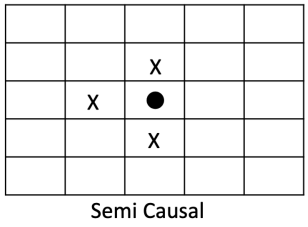
In 2D, a non-causal model can be generalized as follows:

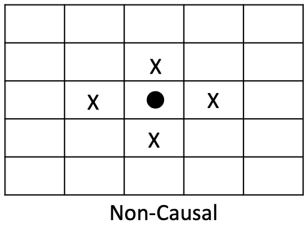
$$f(s) = \sum_{r \in N} \theta_{rf} \theta_r f(s+r) + \sqrt{\beta} w(s) \tag{298}$$

s is a 2D variable in the image plan ($s \in \Omega$). $w(s)$ is the noise random variable with 0 mean and unit variance. Depending on our selection of the neighborhood N, we can either have a causal, non-symmetric half plane, semi-causal, or non-causal model, as shown below. Note the circle denotes the pixel being modeled, while each "x" denotes the known data.





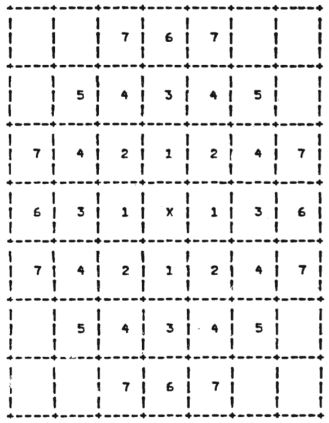




A 2D non-casual model takes the following form:

$$p(f(s)| \text{ all } f(r), s \neq r) = p(f(s)|f(s+r), r \in N) \tag{299}$$

where N is the non-causal neighborhood. This is the definition of a 2D non-causal Markov Random Field model. Consider the Gaussian case. As N increases, we increase the order of the model, as shown below:



The model is represented as follows:

$$f(s) = \sum_{r \in N_s} (f(s+r) + f(s-r)) + e(s) \tag{300}$$

Here N_s is half of the neighborhood of N. For instance, in the first order case, where $N = \{(0,1,(0,-1),(-1,0),(1,0)\}$, $N_s = \{(0,1),(1,0)\}$. Recall that $e(s)$ is the correlated noise with structure:

$$E[e(s)f(r)] = \begin{cases} 0, & r \neq s \\ \nu, & r = s \end{cases} \tag{301}$$

This is the representation of a 2D Gaussian Markov Random Field. For a 2D model, we need to specify boundary conditions. We can represent the image on a "donut." This implies periodic boundary conditions. We generate a vector $\mathbf{f}$ as:

$$\mathbf{f} = \begin{bmatrix} f(0,0) \\ \vdots \\ f(N-1,N-1) \end{bmatrix} \tag{302}$$

where $\mathbf{f}$ is an $N^2 \times 1$ vector. We can construct a 2D GMRF model as follows:

$$B(\boldsymbol{\theta})\mathbf{f} = \mathbf{e} \tag{303}$$

where $\mathbf{e}$ is the correlated noise structure, which is arranged similar to $\mathbf{f}$. Out synthesis equation is therefore simply:

$$\mathbf{f} = B^{-1}(\boldsymbol{\theta})\mathbf{e} \tag{304}$$

Here, $B^{-1}(\boldsymbol{\theta})$ is an $N^2 \times N^2$ matrix. Under periodic boundary conditions, also known as a toroidal lattice structure, $B(\boldsymbol{\theta})$ is a Block-circulant matrix with the following form:

$$B(\boldsymbol{\theta}) = \begin{bmatrix} B_{0,0} & \cdots & B_{0,N-1} \\ B_{0,N-1} & \cdots & B_{0,N-2} \\ B_{0,N-2} & \cdots & B_{0,N-3} \\ \vdots & \ddots & \vdots \\ \cdots & \cdots & B_{0,0} \end{bmatrix} \tag{305}$$

A circulant matrix involves cyclic shifts of the first roow. Circulant matrices can be diagnolized by taking by a discrete Fourier transform (P.J. Davis, Circulant Matrices). We require, that for stability, $||\boldsymbol{\theta}|| < 1$. As a result, for stability,

$$\mu_s = (1 - 2\boldsymbol{\theta}^T\boldsymbol{\phi}_s) > 0, \quad \forall\ s \in \Omega \tag{306}$$

where

$$\phi_s = \text{col}\left[\cos\frac{2\pi}{N}\mathbf{s}^T r\right], \quad r \in N_s \tag{307}$$

Suppose we define a column vector q_s where

$$q_s = \text{col}[1, \lambda_i, \lambda_i^2 t - j, ..., \lambda_i^{m-1} t_j] \tag{308}$$

where s is the spatial variable (i,j) and

$$t_j = \text{col}[1, \lambda_j, \lambda_j^2, ..., \lambda_j^{N-1}], \quad \lambda_i = \exp\left(\sqrt{-1}\frac{2\pi i}{N}\right) \tag{309}$$

Note that effectively the eigenvalues are the Fourier transform of the first row of the Block Circulant matrix and q_s is the inverse 2D DFT. The image vector $\mathbf{f}$ is therefore

$$\mathbf{f} = \frac{1}{N^2}\sum_{s\in\Omega}\frac{\mathbf{q}_s\lambda_s}{\mu_s} \tag{310}$$

where μ_s the eigenvalues. Thus

$$x_s = \mathbf{q}_s^T \mathbf{e} \tag{311}$$

$\mathbf{e}$ is the noise vector and $\mathbf{q}_s^T$ is the forward 2D discrete Fourier transform (DFT). Thus, we require two 2D DFTs to get the image back, a result of the periodic boundary conditions. In summary, GMRFs can be used for texture synthesis, classification, and segmentation, though they were not able to handle discontinuities. Geman and Geman introduced simulated annealing to deal with this. They introduced 2 layers, one for intensity and a second for discontinuities, known as a line process.

Modeling Image Analysis Problems Using Markov Random Fields

Markov Random Fields (MRFs) allows for modeling context dependent entities such as image pixels and correlated features. The practical use of these models results from the Hammersley-Clifford theorem, which sets up an equivalence between MRFs and the Gibbs Distribution. MRF theory allows for modeling the maximum a posteriori (MAP) estimator of contextual dependent patterns, with objective function as the joint posterior probability of the observed data. MRF models are treated as unordered. Let $\mathcal{S}$ be the indices of a discrete set of m sites. In a 2D lattice $\mathcal{S} = \{(i,j)|1 \leq i,j \leq n\}$, where the image is of size $n \times n$. These sites are treated as un-ordered and we can re-index with $k \in \{1,...,m\}$, where $m = n \times n$. Let $\mathcal{L}$ be the set of labels, which are events that may happen to a site. Each set may be either continuous or discrete. For the case of pixel ranges, $\mathcal{L}_c = [X_l, X_h] \subset \mathbb{R}$, though in general $\mathcal{L}_c \subset \mathbb{R}^{a \times b}$. In the discrete case, $\mathcal{L} = \{1,...,M\}$ (e.g. edge, non-edge). Label sets also have ordering (e.g. pixel intensities).

The labeling problem involves assigning a label from $\mathcal{L}$ to a pixel in $\mathcal{S}$. For instance, we can assign a label f_i from $\mathcal{L} = \{$ edge, non-edge $\}$ to each site i. When each site is assigned a unique label, $f_i = f(i)$ is a function with domain $\mathcal{S}$ and image $\mathcal{L}$. In other words

$$f : \mathcal{S} \to \mathcal{L} \tag{312}$$

is a mapping. In random fields, this mapping is known as a configuration, which corresponds to an image, edge map, or some other interpretation of image features. When all sites have the same label set $\mathcal{L}$, then $\mathbb{F} = \mathcal{L}^m$ is the configuration space. In other cases, all labels may not admissible (i.e. the site cannot be interpreted as one of the labels in the label set), which further complicates the problem, as the configuration space is the cross product of each individual label set. In general, there are 4 categories of labeling problems:

(1) LP1: Regular sites with continuous labels (e.g. image smoothing)
(2) LP2: Regular sites with discrete labels (e.g. image restoration, segmentation, edge detection)
(3) LP3: Irregular sites with discrete labels (e.g. perceptual grouping)
(4) LP4: Irregular sites with continuous labels (e.g. pose estimation)

By introducing contextual constraints, we are introducing conditional independencies $P(f_i|\{f_{i'}\})$. In the case of independence (e.g. no context), then

$$P(f) = \prod_{i \in \mathcal{S}} P(f_i) \implies P(f_i|\{f_{i'}\}) = P(f_i) \tag{313}$$

However, as labels are not mutually independent in general, we require Markov random fields to fully solve the problem.

Markov Random Fields and Gibbs Distributions. Let us define a neighborhood $\mathcal{N}$, where

$$\mathcal{N} = \{\mathcal{N}_i | \forall\ i \in \mathcal{S}\} \tag{314}$$

where $\mathcal{N}_i$ is the set of sites neighboring i. This relationship has the following properties:

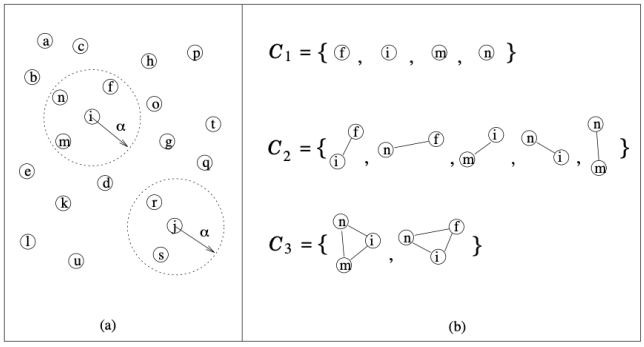
(1) A site is not a neighbor to itself
(2) The neighboring relationship is mutual: $i \in \mathcal{N}_{i'} \iff i' \in \mathcal{N}_i$

We can define the neighborhood as the set of pixels withing radius $\sqrt{r}$ in an image lattice. As discussed above, the order of the neighborhood is proportional to this difference (e.g. first order - four-neighborhood distance, etc). If the set $\mathcal{S}$ is known, we can explicitly define these neighborhood (e.g. for $\mathcal{S} = \{(i,j)|1 \leq i,j \leq n\}$ the first order neighborhood is $\mathcal{N} = \{(i-1,j),(i+1,j),(i,j-1),(i,j+1)\}$). Neighborhoods can also be defined for irregular sites.

The pair $(\mathcal{S},\mathcal{N}) \triangleq \mathcal{G}$ constitutes a graph, where $\mathcal{S}$ are the nodes and $\mathcal{N}$ the edges. A clique c for $(\mathcal{S},\mathcal{N})$ is defined as a subset of sites in $\mathcal{S}$ and can consist of a single site, a pair or sites, or more:

$$C_j = \{\{i_1, i_2, .., i_j\} | i_j \in \mathcal{S} \text{ are neighbors to one another}\} \tag{315}$$

Sites in a clique are ordered as well and the collection of cliques is $\mathcal{C} = C_1 \cup C_2 \cup \cdots$. A representation of neighborhoods and cliques is shown below.



Markov Random Fields: Let $F = \{F_1, .., F_m\}$ be a family of random variables defined on $\mathcal{S}$, where each F_i takes value $f_1 \in \mathcal{L}$. $F = f$ corresponds to the configuration of F in which there is realization of the field. $P(f_i)$ is the associated probability for a discrete field. F is said to be a Markov random field on $\mathcal{S}$ with respect to a neighborhood system $\mathcal{N}$ if

$$P(f) > 0, \quad \forall\ f \in \mathbb{F} \quad \text{positivity} \tag{316}$$

$$P(f_i | f_{\mathcal{S}-\{i\}}) = P(f_i | f_{\mathcal{N}_i}) \quad \text{Markovianity} \tag{317}$$

where $\mathcal{S} - \{i\}$ is the set difference and $f_{\mathcal{S}-\{i\}}$ denotes the corresponding set of labels:

$$f_{\mathcal{N}_i} = \{f_{i'} | i' \in \mathcal{N}_i\} \tag{318}$$

In MRFs, only neighboring labels have direct interactions with each other. If we choose the largest neighborhood in which the neighbors of any sites include all other sites, then any F is an MRF with respect to such a neighborhood system. An MRF is homogeneous if $P(f_i|f_{\mathcal{N}_i})$ is independent of the relative location of the site i in $\mathcal{S}$. In this case, if $f_i = f_j$ and $f_{\mathcal{N}_i} = f_{\mathcal{N}_j}$, then $P(f_i|f_{\mathcal{N}_i}) = P(f_j|f_{\mathcal{N}_j})$. In some problems, we require coupled MRFs, which are coupled via a conditional probability (e.g. one MRF $\{f_i\}$ and another $\{l_{i,i'},\}$ has conditional probability $P(f_i|f_{i'}, l_{i,i'})$.

We now review Markov processes to better understand MRFs. Recall that an nth order Markov Process (MP) is defined as

$$P(f_i|..., f_{i-2}, f_{i-1}) = P(f_i|f_{i-1}, ..., f_{i-n}) \tag{319}$$

A bilateral or non-causal MP depends not only on the past but also on the future. An nth order bilateral MP satisfies

$$P(f_i|..., f_{i-2}, f_{i-1}, f_{i+1}, f_{i+2}, ...) = P(f_i|f_{i+n}, ..., f_{i+1}, f_{i-1}, ..., f_{i-n}) \tag{320}$$

MRFs can be specified in terms of conditional probabilities or joint probabilities. To determine a conditional distribution for an MRF, we use a theoretical result about the equivalence between MRFs and Gibbs distributions,

which provides a mathematically tractable means of specifying the joint probability of an MRF. A set of random variables F is said to be a Gibbs random field (GRF) on $\mathcal{S}$ with respect to $\mathcal{N}$ if and only if its configurations obeys a Gibbs distribution:

$$P(f) = Z^{-1} \times e^{-\frac{1}{T}U(f)} \tag{321}$$

where

$$Z = \sum_{f \in \mathbb{F}} e^{-\frac{1}{T}U(f)} \tag{322}$$

Here Z is a normalizing constant called the *partition function*, T is a constant called the temperature which shall be assumed to be 1 unless otherwise stated, and $U(f)$ is the energy function. The energy

$$U(f) = \sum_{c \in \mathcal{C}} V_c(f) \tag{323}$$

is a sum of clique potential $V_c(f)$ over all possible cliques $\mathcal{C}$. The value of $V_c(f)$ depends on the local configuration on the clique c. The Gaussian distribution is a special member of this Gibbs distribution family.

A GRF is said to be homogeneous if $V_c(f)$ is independent of the relative position of the clique c in $\mathcal{S}$. It is said to be isotropic if V_c is independent of the orientation of c. It is considerably simpler to specify a GRF distribution if it homogeneous or isotropic than one without such properties. To calculate a Gibbs distribution, it is necessary to evaluate the partition function Z, which is the sum over all possible configurations in $\mathbb{F}$. Since there area combinatorial number of elements in $\mathbb{F}$, this is computational challenging. $P(f)$ measures the probability of the occurrence of a particular configuration of pattern f. The more probable configurations are those with lower energies. The temperature T controls the sharpness of the distribution. When the temperature is high, all configurations tend to be equally distributed. Near the zero temperature, the distribution concentrates around the global energy minima. Given T and $U(f)$, we can generate a class of patterns by sampling the the configuration space according to $P(f)$.

The clique potential $V_c(f)$ can be specified by parameters. Sometimes, it may be convenient to express the energy of a Gibbs distribution as the sum of several terms, each ascribed to cliques of a certain size, that is,

$$U(f) = \sum_{\{i\} \in C_1} V_1(f_i) + \sum_{\{i,i'\} \in C_2} V_2(f_i, f_{i'}) + \sum_{\{i,i',i''\} \in C_3} V_3(f_i, f_{i'}, f_{i''}) + ... \tag{324}$$

This implies a homogeneous Gibbs distribution as V_i are independent of the locations. Non non-homogeneous distributions, we require $V_i(i, f_i), V_2(i, i', f_i, f_{i'})$, and so on. If we consider only clicks of size of 2, then

$$U(f) = \sum_{i \in \mathcal{S}} V_1(f_i) + \sum_{i \in \mathcal{S}} \sum_{i' \in \mathcal{N}_i} V_2(f_i, f_{i'}) \tag{325}$$

For this case, the conditional distribution takes the form

$$P(f_i | f_{\mathcal{N}_i}) = \frac{\exp\Big(- \Big[V_1(f_i) + \sum_{i' \in \mathcal{N}_i} V_2(f_i, f_{i'}) \Big] \Big)}{\sum_{f_i \in \mathcal{L}} \exp\Big(- \Big[V_1(f_i) + \sum_{i' \in \mathcal{N}_i} V_2(f_i, f_{i'}) \Big] \Big)} \tag{326}$$

Markov-Gibbs Equivalence: An MRF is characterized by its local property (the Markovianity) whereas a GRF is characterized by its global property (the Gibbs distribution). The Hammersley-Clifford theorem establishes the equivalence of these two types of properties. The theorem states that F is an MRF on $\mathcal{S}$ with respect to $\mathcal{N}$ if and only if F is a GRF on on $\mathcal{S}$ with respect to $\mathcal{N}$.

Proof: Let $P(f)$ be a Gibbs distribution on $\mathcal{S}$ with respect to the neighborhood $\mathcal{N}$. The conditional probability $P(f_i | f_{S-\{i\}})$ is:

$$P(f_i | f_{S-\{i\}}) = \frac{P(f_i, f_{S-\{i\}})}{P(f_{S-\{i\}})} = \frac{P(f)}{\sum_{f'_i \in \mathcal{L}} P(f')} \tag{327}$$

where $f' = \{f_1, ..., f_{i-1}, f'_i, ..., f_m\}$. We can write $P(f)$ in terms of its clique potentials as

$$P(f_i|f_{S-\{i\}}) = \frac{\exp\Big(-\sum_{c\in\mathcal{C}} V_c(f)\Big)}{\sum_{f_i'}\exp\Big(-\sum_{c\in\mathcal{C}} V_c(f)\Big)} \tag{328}$$

Next we divide $\mathcal{C}$ into two sets $\mathcal{A}$ and $\mathcal{B}$ with $\mathcal{A}$ consisting of cliques containing i and $\mathcal{B}$ consisting of cliques not containing i. Then

$$P(f_i|f_{S-\{i\}}) = \frac{\Big[\exp\Big(-\sum_{c\in\mathcal{A}} V_c(f)\Big)\Big]\Big[\exp\Big(-\sum_{c\in\mathcal{B}} V_c(f)\Big)\Big]}{\sum_{f_i'}\Big[\exp\Big(-\sum_{c\in\mathcal{A}} V_c(f)\Big)\Big]\Big[\exp\Big(-\sum_{c\in\mathcal{B}} V_c(f)\Big)\Big]} \tag{329}$$

Because $V_c(f) = V_c(f')$ for any clique that does not contain i (i.e. MRF), then the above reduces to:

$$P(f_i|f_{S-\{i\}}) = \frac{\exp\Big(-\sum_{c\in\mathcal{A}} V_c(f)\Big)}{\sum_{f_i'}\exp\Big(-\sum_{c\in\mathcal{A}} V_c(f)\Big)} \tag{330}$$

This proves that a GRF is an MRF. The converse proof is more difficult and is not completed here. This theorem allows for specification of the joint probability distribution by specifying clique potentials. Parameter estimation and choosing the forms of clique potentials is known as MRF modeling. Note that to calculate the joint probability of an MRF, then we require the partition function. While this is avoided in maximum-probability based models, in general, this remains a difficult task.

There exists a unique normalized potential, called the canonical potential, for every MRF. Let $\mathcal{L}$ be a countable label set. $V_c(f)$ is normalized if $V_c(f) = 0$ for $i \in c$ that take on a specific label (e.g. $\mathcal{L} = \{0, 1, ..., M\}$. Let F be a random field on $\mathcal{S}$ with characteristics $P(f_i|f_{\mathcal{S}-\{i\}}) = P(f_i|f_{\mathcal{N}_i})$. F is a Gibbs field with canonical potential function defined as

$$V_c(f) = \begin{cases} 0, & c = \phi \\ \sum_{b\subset c}(-1)^{|c-b|}\ln P(f_b), & x \neq \phi \end{cases} \tag{331}$$

where ϕ is the empty set, $|c-b|$ is the number of elements in the set $c-b$ and $f_i^b = f_i$ if $i \in b$ (configuration which agrees with f on b but assigns 0 outside of b. This potential is

$$V_c(f)\sum_{b\subset c}(-1)^{|b-c|}\ln P(f_i^b|f_{\mathcal{N}_i}^b) \tag{332}$$

The canonical potential can be written if $P(f)$ is known, which is not the case. Nonetheless, there is an indirect way: Use a non-canonical representation to derive $P(f)$ and then canonicalize it to obtain the unique canonical representation.

Auto-Models. Contextual constraints on two labels are the lowest order constraints to convey contextual information. They are widely used because of their simple form and low computational cost. They are encoded in the Gibbs energy as pair-site clique potentials. With clique potentials of up to two sites, the energy takes the form

$$U(f) = \sum_{i\in\mathcal{S}} V_1(f_i) + \sum_{i\in\mathcal{S}}\sum_{i'\in\mathcal{N}_i} V_2(f_i, f_{i'}) \tag{333}$$

We require a method to select V_i. First let us consider the case when $V_1(f_i) = f_i G_i(f_i)$ and $V_2(f_i, f_{i'}) = \beta_{i,i'} f_i f_{i'}$. Here $G(\cdot)$ is an arbitrary function and $\beta_{i,i'}$ represents pair-site interactions. In this case, the energy is

$$U(f) = \sum_{i\in\mathcal{S}} f_i G_i(f_i) + \sum_{i\in\mathcal{S}}\sum_{i'\in\mathcal{N}_i} \beta_{i,i'} f_i f_{i'} \tag{334}$$

These are known as auto-models. If f_i takes on values in $\mathcal{L} = \{0, 1\}$, then

$$U(f) = \sum_{\{i\} \in C_1} \alpha_i f_i + \sum_{\{i,i'\} \in C_2} \beta_{f_i, f_{i'}} f_i f_{i'} \tag{335}$$

This is known as an auto-logistic model. In the case of the 4-nearest neighbors on a lattice, then we have the Ising model. The conditional distributions for the auto-logistic model is:

$$P(f_i | f_{\mathcal{N}_i}) = \frac{\exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)}{\sum_{f_i \in \{0,1\}} \exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)} = \frac{\exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)}{1 + \exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)} \tag{336}$$

An auto-model is auto-bionomial if $f_i = \{0, 1, ...m_1\}$ and f_i has a conditionally bionomial distribution:

$$P(f_i | f_{\mathcal{N}_i}) = \binom{M-1}{f_i} q^{f_i} (1-q)^{M-1-f_i}, \quad q = \frac{\exp\left(\alpha_i + \sum_{i \ \in \mathcal{N}_i} \beta_{i,i'} f_{i'}\right)}{1 + \exp\left(\alpha_i + \sum_{i \ \in \mathcal{N}_i} \beta_{i,i'} f_{i'}\right)} \tag{337}$$

The corresponding energy function is

$$U(f) = -\sum_{\{i\} \in C_1} \ln \binom{M-1}{f_i} - \sum_{\{i\} \in C_1} \alpha_i f_i - \sum_{i,i' \in C_2} \beta_{f_i, f_{i'}} f_i f_{i'} \tag{338}$$

An auto-normal model is also known as a Gaussian MRF with conditional PDF

$$P(f_i | f_{\mathcal{N}_i}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \left[f_i - \mu_i - \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} (f_{i'} - \mu_{i'}) \right]^2 \right) \tag{339}$$

with mean and variance

$$E(f_i | f_{\mathcal{N}_i}) = \mu_i - \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} (f_{i'} - \mu_{i'}) \tag{340}$$

$$\text{var}(f_i | f_{\mathcal{N}_i}) = \sigma^2 \tag{341}$$

The joint distribution is a Gibbs distribution with

$$p(f) = \frac{\sqrt{\det B}}{\sqrt{(2\pi\sigma^2)^m}} \exp\left(\frac{(f-\mu)^T B (f-\mu)}{e\sigma^2} \right) \tag{342}$$

Here f is an $m \times 1$ vector, μ is an $m \times 1$ vector of conditional means, and $B = [b_{i,i'}]$ is the $m \times m$ interaction matrix with diagonal elements 0 and off diagonal elements are $-\beta_{i,i'}$. Therefore, the clique potentials are:

$$V_1(f_i) = \frac{(f_i - \mu_i)^2}{2\sigma^2} \tag{343}$$

$$V_2(f_i, f_{i'}) = \frac{\beta_{f_i, f_{i'}} (f_i - \mu_i)(f_{i'} - \mu_{i'})}{2\sigma^2} \tag{344}$$

A field of independent Gaussian noise is a special MRF whose Gibbs energy consists of only single-site clique potentials. Because all higher order clique potentials are zero, there is no contextual interaction in the independent Gaussian noise. Note that B is the inverse of the covariance matrix.

A simultaneous atuo-regression model (SAR model) involves a set of m equations

$$f_i = \mu_i + \sum \beta_{i,i'} (f_{i'} - \mu_{i'}) + e_i \tag{345}$$

where e_i are indepednent Gaussian with 0 mean and variance σ^2. This will generate class of all multivariate normal distributions but with joint p.d.f. as

$$p(f) = \frac{\det(B)}{\sqrt{(2\pi\sigma^2)^m}} \exp\Big(\frac{(f-\mu)^T B^T B(f-\mu)}{2\sigma^2}\Big) \tag{346}$$

The Labeling Problem - Introduction

We will be focusing on the labeling problem (e.g. segmentation, edge detection, etc.), which involves the functional mapping from the image set to the label set. Several estimators, including the MAP estimator, which requires knowledge of the posterior, prior (smoothness criteria), and likelihood. The posterior can be non-convex in many applications, which results in difficulties in optimization. The method of simulated annealing is one method to overcome this. Once we have a MRF, we have a sampling problem, which will be discussed in detail.

Markov Random Fields

Let us consider a 1D MRF and a 2D non-causal MRF. Suppose we have the following joint density

$$p(f(1),..,f(N)) = p(f(N)|f(N-1)...f(1))p(f(N-1)...f(1)) \tag{347}$$

For an mth order 1D markov process, then

$$\begin{aligned} p(f(1),..,f(N)) &= p(f(N)|f(N-1)...f(1))p(f(N-1)...f(1)) \\ &= p(f(N)|f(N-1)...f(N-m))p(f(N-1)...f(1)) \\ &= p(f(N)|f(N-1)...f(N-m))p(f(N-1)|f(N-2),...f(1))p(f(N-2),...f(1)) \\ &= p(f(N)|f(N-1)...f(N-m))p(f(N-2),...f(1))p(f(N-1)|f(N-2),...f(N-1-m)) \\ &= ... \end{aligned} \tag{348}$$

In other words, we can write this field as a product of local conditional densities. Let us assume that $p(f(i)|f(i-1)...f(i-m-1))$ Gaussian, then one possible model is a time series model. So we can write

$$f(n) = \sum_{j=1}^{m} \theta_j f(n-j) + \sqrt{\rho} W(n) \tag{349}$$

where $W(n)$ is a zero mean Gaussian. If we stack all the $f(i)$ as a vector

$$\mathbf{f} = \begin{bmatrix} f(1) \\ \vdots \\ f(n) \end{bmatrix} \tag{350}$$

then

$$B(\boldsymbol{\theta})\mathbf{f} = \sqrt{\rho}\mathbf{W} = \sqrt{\rho} \begin{bmatrix} W(1) \\ \vdots \\ W(n) \end{bmatrix} \tag{351}$$

Th mean vector of $\mathbf{f}$, $E[\mathbf{f}] = 0$. As $\mathbf{f} = \sqrt{\rho}B^{-1}(\boldsymbol{\theta})\mathbf{W}$, and as $E[\mathbf{W}\mathbf{W}^{\mathbf{T}}] = I$ (Gaussian white noise) the covariance matrix is

$$E[\mathbf{f}\mathbf{f}^T] = \rho B^{-1}(\boldsymbol{\theta}) B^{-1^T}(\boldsymbol{\theta}) \tag{352}$$

Now the conditional distribution is

$$p(\boldsymbol{f}|\boldsymbol{\theta}) = \mathcal{N}(0, \rho B^{-1}(\boldsymbol{\theta}) B^{-1^T}(\boldsymbol{\theta})) \tag{353}$$

The local and global transformation is simply a transformation of random variables. Similar arguments can be made for 2D causal methods and non symmetric half plane representations.

The labeling problem is formulated as follows. Suppose we have m sites (m pixels in the image) and a label set $\mathcal{L} = \{0, 1\}$ (e.g. edge, non edge), or one of several classes. The configuration space is a Cartesian product

$$\mathbb{F} = \mathcal{L} \times \mathcal{L} \cdots = \mathcal{L}^m \tag{354}$$

In the case of m pixels and M labels, we have M^m possible configurations (or realizations). This is a difficult problem considering the size of an image. If all the pixels are independent, then for $\mathbf{f} = [f(1), ..., f(n)]^T$ then

$$p(\mathbf{f}) = \prod_{i \in \mathcal{S}} p(f_i) \tag{355}$$

where $\mathcal{S}$ is the image array. Now let us consider pixels (and sites) (i.e. the picture is ordered). Here $\mathcal{S}$ is the image space (graph, etc.). Let N be the neighborhood $\{N_i | \ \forall i \in \mathcal{S}\}$. N_i is the set of sites neighboring i. A site is not a neighbor to itself ($i \notin N_i$). Furthermore, sites are mutual $i \in N_{i'} \implies i' \in N_i$. For a regular lattice structure, then

$$N_i = \{i' \in \mathcal{S} | \text{dist(pixel i',pixel i)} \leq r, i \neq i'\} \tag{356}$$

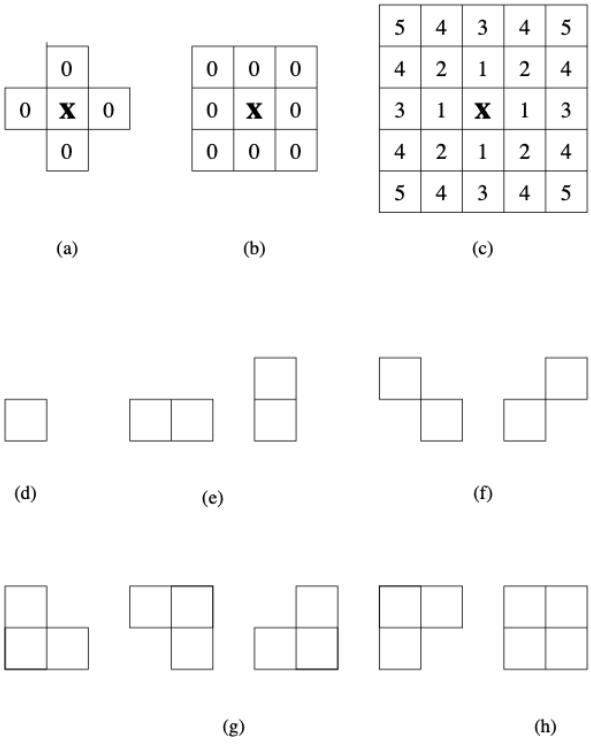
A pair $(\mathcal{S}, N)$ is a graph, where $\mathcal{S}$ contains the nodes, and N the links between the nodes. A clique for c for $(\mathcal{S}, N)$ is a subset of sites in $\mathcal{S}$ and could be a single site $c = \{i\}$, pair of sites $c = \{i, i'\}$, or a triplet of sites $c = \{i, i', i''\}$. We use the following notation:

$$C_1 = \{i | i \in \mathcal{S}\} \tag{357}$$

$$C_2 = \{(i, i') | i \in \mathcal{S}, i' \in N_i\} \tag{358}$$

$$C_3 = \{(i, i', i'') | i, i', i'' \in \mathcal{S}; i', i'' \in N_i\} \tag{359}$$

Examples of cliques and neighbors are shown below.



Let $\mathbf{f} = [f_1, ..., f_m]^T$, then

$$p(\mathbf{f}) \geq 0 \tag{360}$$

$\mathcal{F}$ is a set of random variables defined on $\mathcal{S}$. $\mathcal{F} = \{\mathcal{F}_1, ..., \mathcal{F}_m\}$. f_i is the value taken by the random variable F. Next, we know that

$$p(f_i|f_{\mathcal{S}-\{i\}}) = p(f_i|f_{N_i}) \tag{361}$$

This is the conditional distribution. In the case of 2D MRFs, it is difficult to go from local to conditional, though we can marginalize to get from global to local. Recall that for a causal model

$$p(f_i|...) = p(f_i|f_{i-1}, ..., f_{i-n}) \tag{362}$$

For a bilateral/non causal model, we have

$$p(f_i|...) = p(f_i|f_{i+n}, ..., f_{i-n}) \tag{363}$$

The Gibbs distribution takes the form:

$$p(\mathbf{f}) = \frac{1}{Z} e^{-U(f)/T} \tag{364}$$

where

$$Z = \sum_{f \in \mathcal{F}} e^{-U(f)/T} \tag{365}$$

$U(f)$ is the energy function, whose structure is determined by cliques. T is the temperature. We reduced T with an annealing schedule to obtain the optimal posterior. Note that

$$U(f) = \sum_{c \in C} V_c(f) \tag{366}$$

$V_c(f)$ is known as the clique potentials, and c is the subset of cliques. We can capture the local dependencies of the pixels:

$$U(f) = \sum_{\{i\} \in C_1} V_1(f_i) + \sum_{\{i,i'\} \in C_2} V_2(f_i, f_{i'}) + \sum_{\{i,i',i''\} \in C_3} V_3(f_i, f_{i'}, f_{i''}) \tag{367}$$

Consider the case, where we consider pairs of cliques. Then,

$$U(f) = \sum_{i \in \mathcal{S}} V_1(f_i) + \sum_{i \in \mathcal{S}} \sum_{i' \in \mathcal{N}_i} V_2(f_i, f_{i'}) \tag{368}$$

The utility of the Gibbs distribution in MRFs is due to the Hammersley Clifford Theorem, known as the MRF-Gibbs equivalence. $\mathcal{F}$ is an MRF on $\mathcal{S}$ with respect to N iff $\mathcal{F}$ is a Gibbs random field on $\mathcal{S}$ with respect to N. Showing the Gibbs to MRF relationship is easy, while the MRF to Gibbs relationship is hard.

Auto Models

Recall that for pairs of cliques have energy function

$$U(f) = \sum_{i \in \mathcal{S}} V_1(f_i) + \sum_{i \in \mathcal{S}} \sum_{i' \in \mathcal{N}_i} V_2(f_i, f_{i'}) \tag{369}$$

The most simple auto model is the Ising model, which will be discussed later. In general, for auto models, $V(f_i) = f_i G_i(f_i)$ and $V_2(f_i, f_{i'}) = \beta_{i,i'} f_i f_{i'}$. Suppose $\mathcal{L} = \{0, 1\}$. Then

$$U(f) = \sum_{\{i\} \in C_1} \alpha_i f_i + \sum_{\{i,i'\} \in C_2} \beta_{f_i, f_{i'}} f_i f_{i'} \tag{370}$$

Here $\beta_{i,i'}$ are the interaction coefficients. These are difficult due to the non-causal representation in two dimensions. The local conditional distribution here is simply

$$P(f_i|f_{\mathcal{N}_i}) = \frac{\exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)}{\sum_{f_i \in \{0,1\}} \exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)} = \frac{\exp\left(\alpha_i f_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_i f_{i'}\right)}{1 + \exp\left(\alpha_i + \sum_{i' \in \mathcal{N}_i} \beta_{i,i'} f_{i'}\right)} \tag{371}$$

This is known as the Ising Model. For a Gaussian energy function, we have

$$P(f_i|f_{\mathcal{N}_i}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(- \frac{1}{2\sigma^2} \left[f_i - \mu_i - \sum_{i' \in \mathcal{N}_i} \beta_{i,i'}(f_{i'} - \mu_{i'}) \right]^2 \right) \tag{372}$$

with mean and variance

$$E(f_i|f_{\mathcal{N}_i}) = \mu_i - \sum_{i' \in \mathcal{N}_i} \beta_{i,i'}(f_{i'} - \mu_{i'}) \tag{373}$$

$$\text{var}(f_i|f_{\mathcal{N}_i}) = \sigma^2 \tag{374}$$

The joint distribution is a Gibbs distribution with

$$p(f) = \frac{\sqrt{\det B}}{\sqrt{(2\pi\sigma^2)^m}} \exp\left(\frac{(f-\mu)^T B (f-\mu)}{2\sigma^2}\right) \tag{375}$$

Here f is an $m \times 1$ vector, μ is an $m \times 1$ vector of conditional means, and $B = [b_{i,i'}]$ is the $m \times m$ interaction matrix with diagonal elements 0 and off diagonal elements are $-\beta_{i,i'}$. Note that this is identical to the GMRF representation identified above if $\mu_i = 0$.

Image Synthesis

We desire to synthesize a binary texture (e.g. values $(0, 1)$). As a result, we will utilize the Ising model with α, β given. To synthesis this texture, we need to do sampling from $p(\mathbf{f})$. One method for sampling is Metropolis Sampling.

Metropolis Sampling: The algorithm for metropolis sampling is:

(1) Randomly initialize $\mathbf{f}$
(2) Define our image lattice $\mathcal{S}$
(3) For $i \in \mathcal{S}$:
  (a) Let $f'_{i'} = f_{i'}$ for all $i' \neq i$
  (b) Choose $f_i \in \mathcal{L}$ at random.
  (c) Let $p = \min\{1, p(f')/p(f)\}$
  (d) Replace f by f' with probability p
  (e) Generate a uniform random variable $u \in (0, 1)$.
  (f) If $u \leq p$ replace f by f'. If $u > p$, no change.

Gibbs Sampling. Both Metropolis and Gibbs sampling are special cases of MCMC (Monte Carlos Markov Chain) sampling algorithms. For the Gibbs sampler, we have the following algorithm:

(1) Let $\mathbf{f}$ be a random configuration
(2) For $i \in \mathcal{S}$, do:
  (a) Compute $p_l = p(f_i = 1|f_{N_i})$ $\forall l \in \mathcal{L}$, where f_{N_i} are the pixel values at neighboring sites.
  (b) Set f_i to l with probability p (this is performed by drawing from a uniform and deciding whether or not to replace it).

MAP-MRF Formulation

Let $p(\mathbf{f})$ be the calculated posterior density function. We desire an f but we have d. The likelihood $p(d|f)$ is known. We desire the MAP (maximum a posterior) estimator that $\mathbf{f}$ that maximizes $p(\mathbf{f}|d)$. From Bayes rule

$$\max_{\mathbf{f}} p(\mathbf{f}|\mathbf{d}) = \max_{\mathbf{f}} \frac{p(\mathbf{d}|\mathbf{f})p(\mathbf{f})}{\mathbf{d}} = \max_{\mathbf{f}} p(\mathbf{d}|\mathbf{f})p(\mathbf{f}) \tag{376}$$

Here, $p(\mathbf{d}|\mathbf{f})$ is Gaussian noise and $p(\mathbf{f})$ is an MRF, GMRF, or GRF. For a simple model

$$\mathbf{d} = \mathbf{f} + \boldsymbol{\eta} \tag{377}$$

where $\boldsymbol{\eta}$ is Gaussian noise, then the MAP-MRF formulation yielded simple results. In some applications, $\mathbf{d}$ is the convolution

$$\mathbf{d} = \mathbf{h} * \mathbf{f} + \boldsymbol{\eta} \tag{378}$$

$\mathbf{h}$ is a blur function (e.g. relative motion, etc.). In this case, $p(\mathbf{f}|\mathbf{d})$ has multiple local and global maxima as it is non-convex. To solve this optimization problem, we need our optimizer to worsen the cost function in some instances with probability p, which is essential to simulated annealing.

Texture Segmentation

Let Ω be the set of grid points in an $M \times M$ lattice. Then we can construct a model with labels L_s and zero mean gray level arrays Y_s. We can write the following expression for the conditional density for the pixel:

$$P(Y_s = y_s | Y_r = y + r, r \in N_s, L_s = l) = \frac{\exp[-U(Y_s = y_s | Y_r = y + r, r \in N_s, L_s = l)]}{Z(l | y_r, r \in N_s)} \tag{379}$$

Note that l is a line process that characterizes the discontinuities in the image. The Gibbs distribution is

$$U(Y_s = y_s | Y_r = y + r, r \in N_s, L_s = l) = \frac{1}{2\sigma_l^2}\left(y_s^2 - 2\sum_{r \in N_s} \Theta_{s,r}^l y_s y_r\right) \tag{380}$$

Using this Gibbs distribution, the joint density in the window

$$P(\mathbf{Y}_s^* | L_s = l) = \frac{\exp[-U(\mathbf{Y}_s^* | L_s = l)]}{Z_L(l)} \tag{381}$$

$$U(\mathbf{Y}_s^* | L_s = l) = \frac{1}{2\sigma^2}\sum_{r \in W_s}\left\{y_r^2 - \sum_{r \in N^* | r+\tau \in W_s} \Theta_\tau^l y_r (y_{r+\tau} + y_{r-\tau})\right\} \tag{382}$$

We also need to model the label array

$$P(L_s | L_r, r \in \hat{N}_s) = \frac{1}{Z_2} e^{-U_2(L_s | L_r)}; \quad U(L_s | L_r, r \in \hat{N}_s) = -\beta \sum_{r \in \hat{N}_s} \delta(L_s - L_r), \quad \beta > 0 \tag{383}$$

Using Bayes rule

$$P(L_s | \mathbf{Y}_s^*, L_r, r \in \hat{N}_s) = \frac{P(\mathbf{Y}_s^* | L_s) P(L_s | L_r, r \in \hat{N}_s)}{P(\mathbf{Y}_s^*)} = \frac{1}{Z_p} \exp[-U_p(L_s | \mathbf{Y}_s^*, L_r, r \in \hat{N}_s)] \tag{384}$$

We can then choose to maximize the entire image

$$P(\mathbf{L} | \mathbf{Y}^*) = \frac{P(\mathbf{Y}^* | \mathbf{L}) P(\mathbf{L})}{P(\mathbf{Y}^*)} \tag{385}$$

Thus for the entire image

$$P \sim \frac{1}{Z_{T_i}} \exp\left[-\frac{1}{T_i} U_p(L_s | \mathbf{Y}_s^*, L_r, r \in \hat{N}_s)\right] \tag{386}$$

Simulated annealing involves a cooling schedule for the temperature to find the MAP estimator, for example,

$$T_k = \frac{T_0}{1 + \log_2 k} \tag{387}$$

where k is the iteration. When the temperature is high, the bond between adjacent pixels is loose, and the distribution tends to behave like a normal distribution. As T_k decreases, the distribution concentrates on lower values of the energy function (i.e. higher probabilities). Geman and Geman shows that this converges.

Summary - MAP Estimation

$p(\mathbf{f})$ is the likelihood distribution for the image and is necessary for the MAP estimate. Calculation of the MAP estimator from the posterior requires simulated annealing, given the non-convex nature of the function (1984 PAMI; Geman and Geman). Recall that

$$p(\mathbf{f}) = \frac{1}{Z} e^{-U(\mathbf{f})\mathbf{f}} \tag{388}$$

where Z is the partition function and the image $f(m-1, m-1)$ is arranged as a vector. $U(\mathbf{f})$ is the energy function in terms of clique potentials. Auto models utilize neighborhoods that contain the 8 nearest neighbors. The simplest case of auto models is the Icing model, when $f_i = (0, 1)$. For texture segmentation, we have two arrays, the label array

$$\mathbf{L} = \begin{pmatrix} l(0,0) \\ \vdots \\ l(m-1, m-1) \end{pmatrix} \tag{389}$$

and the image array $\mathbf{f}$. For the MAP estimator, we desire

$$\max_L p(\mathbf{L}|\mathbf{f}) = \max_{\mathbf{L}} \frac{p(\mathbf{f}|\mathbf{L})p(\mathbf{L})}{p(\mathbf{f})} = \max_{\mathbf{L}} p(\mathbf{f}|\mathbf{L})p(\mathbf{L}) \tag{390}$$

$p(\mathbf{L})$ is itself a prior and is a multi-level model (in the binary case, this is the Icing model). We can write this as a conditional density $p(\mathbf{f}|\mathbf{L})$ as a Gaussian, which is a GMRF. This is the maximum a posteriori method. Furthermore, $p(\mathbf{L}|\mathbf{f})$ is non-convex. To optimize, we use aa method known as simulated annealing, in which we convert the distribution into an energy function. We implement this with the Gibbs sampler (like synthesis the image f given $p(f)$). We want a configuration that maximizes the posterior, which can be determined from a Gibbs sampler.

The Image Restoration Problem

Let

$$\mathbf{g} = H\mathbf{f} + \eta \tag{391}$$

where $\mathbf{g}$ is the degraded image, H is the point spread function, and η is additive Gaussian noise. An estimate of $\mathbf{f}$ is desired. Using the MAP method,

$$p(\mathbf{f}|\mathbf{g}) = \max_f \frac{p(\mathbf{g}|\mathbf{f})p(\mathbf{f})}{p(\mathbf{g})} = \max_f p(\mathbf{g}|\mathbf{f})p(\mathbf{f}) \tag{392}$$

Here, $p(\mathbf{f})$ can be a Gibbs distribution, and $p(\mathbf{g}|\mathbf{f})$ is the noise model. In cases when H is unknown, this is known as blind restoration. Again as $p(\mathbf{f}|\mathbf{g})$ os non-convex, we require simulated annealing to find a solution. All Bayesian formulations can be used with a Gibbs distribution prior and simulated annealing for optimization.

Furthermore, $\mathbf{f}$ is not a stationary model. We therefore require a joint distribution for the line process $p(\mathbf{f}, \mathbf{l})$, where $\mathbf{l}$ is a line process. Similar to labels, we want to estimate $\mathbf{f}$ and $\mathbf{l}$, as if we are synthesising $\mathbf{l}$. In this case, we have

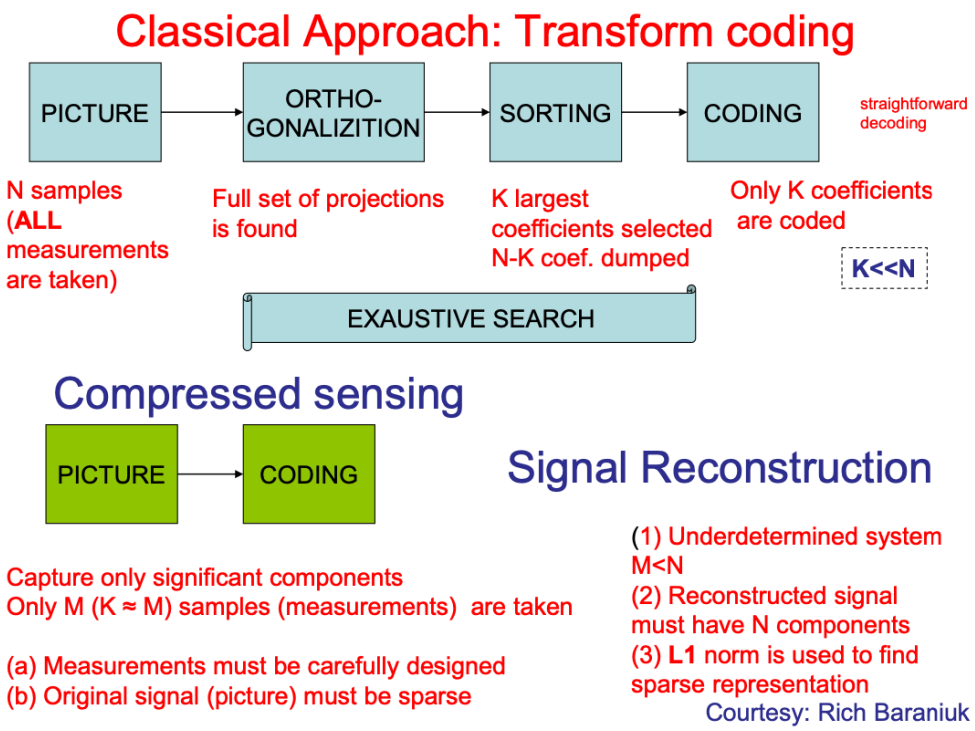
$$p(\mathbf{f}, \mathbf{l}) = \frac{1}{Z} e^{-U(\mathbf{f},\mathbf{l})} \tag{393}$$

$U(\mathbf{f}, \mathbf{l})$ will describe intensities and discontinuities, which will be modeled as linear structures.

Compressive sensing, Sparse representations, and dictionaries with applications in computer vision

For a class of signals that are sparse, it is possible sample at a frequency lower than the Nyquist criteria. Recall that the Shannon/Nyquist sampling theorem states that sampling rate must be at least twice the message signal bandwidth in order to achieve exact recovery. Compressed sensing is a new method to capture and represent compressible signals at the rate well below Nyquist's rate. These methods: (a) employ nonadaptive linear projections (random measurement matrix), (b) preserve the signal structure (length of the sparse vectors is conserved), and (c) reconstruct the signal from the projections using optimization process (L1 norm).

A classical approach for transform coding involves doing a DCT/DST, looking at magnitudes, and invert the process. We assume that the signal is sparse and reconstruct in a certain way, as shown in the image below:



Sparsity for images is not a new concept, with wavelet transforms only useful information in the top left hand corner. In compressive sensing, we directly acquire "compressed" data. We replace samples by more general measurements. Given M (the dimensionality of the acquired data y), we desire a method to determine N (the dimensionality of the true data x). When data is sparse/compressible, can directly acquire a condensed representation with no/little information loss through linear dimensionality reduction:

$$y = \Phi x \tag{394}$$

Since $y \in M \times 1$, and $x \in N \times 1$, then Φ is rectangular. Φ can be coefficients of Fourier series, etc. In general, this is the core compressive sensing problem. This problem is solvable if Φ obeys what is known as the restricted isometry property (RIP). A matrix Θ is said to satisfy the RIP of order K with constants $\delta_k \in (0, 1)$ if

$$(1 - \delta_K)||v||_2^2 \leq ||\Theta v||_2^2 \leq (1 + \delta_K)||v||_2^2 \tag{395}$$

for any v such that $||v||_0 \leq K$. RIP is a sufficient condition to find the sparsest solution. When RIP holds Θ approximately preserves the Euclidian length of K-sparse signals. All subsets of K - columns taken from Θ are nearly orthogonal. RIP has been established for some matrices such as random Gaussian, Hadamard, and Fourier, however, in practice there is no computationally feasible way to check this property for a given matrix, as it is combinatorial in nature. In general, however, this is an NP hard problem. The RIP condition states that the eigenvalues of any subset of at most K rows of the ensemble Φ are at most $\delta_K \in (0, 1)$ removed from 1. This implies that any subset of at most K rows forms a basis for that particular subspace. Thus, an ensemble that

adheres to the RIP forms a basis for any K-sparse signal. Hence enough information is available to reconstruct the original signal, on the condition that a sufficient number of measurements are recorded. Verifying RIP requires $\binom{N}{K}$ (all) possible combinations of K non-zero entries of length N. However, RIP could be achieved by simply selecting Φ randomly. ϕ_{ij} are iid random variables from a Gaussian probability density with zero mean and 1/N varaince. So the measurements $y = \Phi x$ are randomly weighted linear combinations of elements of signal.

For compresssible signals, we can make further simplifications. Let $\mathbf{x}$ be the target signal, representation of an image in time or space with domain $\mathbb{R}^N$:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \tag{396}$$

Let Ψ be an orthonormal basis for $\mathbf{x}$, where

$$\Psi = \begin{bmatrix} \psi_{11} & \cdots & \psi_{1N} \\ \vdots & \ddots & \vdots \\ \psi_{N1} & \cdots & \psi_{NN} \end{bmatrix} \tag{397}$$

The classical approach is to find the signal projections on a given basis

$$x = \sum_{i=1}^{N} s_i \psi_i \tag{398}$$

Here s_i is a representation of the imaging in the Ψ domain and $s_i = \Psi_i^T x$. x is only K-sparse if only K basis vectors ψ_i have $s_i = 0$. In practice, x is compressible if it has just a few large s coefficients and many small s coefficients. Compressible signals are well represented by K-sparse representations. In compressed sensing, we measure only significant components $M < N$). Let $\mathbf{y}$ be measurements and Φ be the measurement matrix. Then $y = \Phi x = \Phi \Psi s \implies y = \Theta s$. We are now constructing x in an orthonormal basis.

Our approach to designing a signal reconstruction algorithm is as follows. Draw Φ from a iid Gaussian or Bernoulli to ensure the RIP property and that they are stable. Note that we have M measurements y, a random measurements matrix Φ, and an orthonormal basis that we select Ψ and can use these to reconstruct a compressible signal x of length N or equivalently its sparse coefficients vector s. Since $M < N$, there are infinitely many solutions. But we have a restriction that the solution is sparse. The signal x has to be reconstructed from the underdetermined system using optimization (norm L1) linear combinations, since the projections(linear combinations are measured) we are extracting the sparse components using optimization algorithm (L1) with restrictions (RIP).

Signal reconstruction algorithm aims to find signal's sparse coefficient vector. The L2 norm (energy) is minimized. While the pseudo inverse is closed form and finds a solution, it DOES NOT find sparse solution. The L0 norm counts the number of non-zero elements of s. This optimization can recover a K dimension sparse signal exactly with high probability using only $M = K + 1$ measurements, but solving it is unstable and NP-complete, requiring exhaustive enumeration of all (NK) possible locations of the non-zero entries in s. L1 norm (adding absolute values of all elements) can exactly recover K dimension sparse signals and closely approximates compressible signals with high probability using only $M \geq cL \log(N/K)$ iid Gaussian measurements. Convex optimization reducing to linear programming is known as Basis pursuit, with computation complexity of about $O(N^3)$. Mathematically, each of these can be written as follows. Given $y = \Phi x$, we can find x, if it is sparse as:

$$\hat{x} = \arg\min_{y=\Phi x} ||x||_2 \tag{399}$$

$$\hat{x} = \arg\min_{y=\Phi x} ||x||_0 \tag{400}$$

$$\hat{x} = \arg\min_{y=\Phi x} ||x||_1 \tag{401}$$

where $||x||_2$, $||x||_1$, and $||x||_0$ represent the l_2, l_1, and l_0 norm, respectively. The l_2 norm is computationally efficient but incorrect. The l_0 norm is correct, but slow. The l_1 norm is correct and efficient, though there

is mild oversampling, and requires only linear programming. Then number of measurements required here is $M = O(K \log(N/K)) << N$. Furthermore, there is l_0 l_1 equivalence in certain conditions (Donoho).

Basis pursuit denoising (compared to what was discussed above, known as basis pursuit/linear programming) involves considering noise into the measured signal. Iterative greedy algorithms, like OMP (orthogonal matching pursuit) is a further improvement. Let us now consider BPDN (Basis Pursuit Denoising). Recall that basis pursuit is a linear program and involves the optimization

$$\min ||x||_1 \quad \text{subject to } y = \Phi x \tag{402}$$

In the case of noisy measurements, we have

$$y = \Phi x + n \tag{403}$$

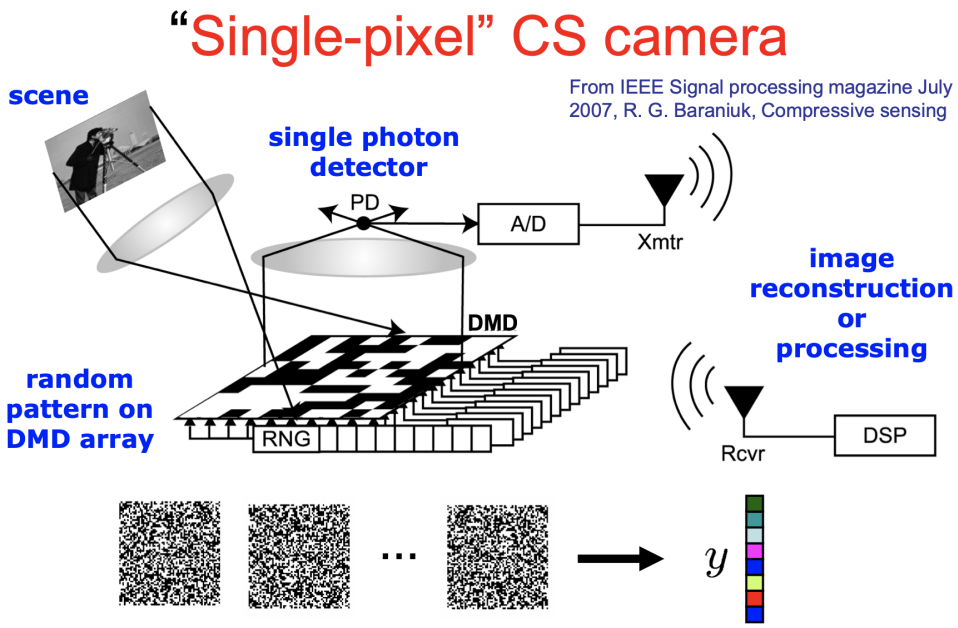
The BPDN algorithm, which is a convex quadratic program, involves the following optimization:

$$\min ||x||_1 \quad \text{such that } ||y - \Phi x||_2 \leq v \tag{404}$$

BPDN is motivated by MRI reconstruction, in order to decrease scan time, which is an example application. Compressive sensing in the context of computer vision involves shape, pose, motion recovery, dense reconstruction from sparse data (depth from gradients, stereo), and many others. In computer vision, data deluge and redundancy is present (e.g. frame differences are sparse). Furthermore, applications in biological perception are also sparse (e.g motion, landmark-based navigation, time to contact, salient pixels, salient views, etc.).

Single Pixel CS Camera

The goal of data acquisition is acquisition of y. Consider the "single-pixel" camera. For this camera, we have the original image, we have an array which randomly flips pixel, analogous to the matrix Φ. From there, we flip the mirror array M times to acquire M measurements and employ sparsity based recovery



Applications of Compressive Sensing

This is particularly useful in the context of a high speed camera (upwards of 10,000 fps). Such cameras have low exposure time and thus very low SNR. We can formulate this as a compressive sensing. We can consider periodic motion in a high speed camera as as a periodic signal $x(t)$ with period P and band-limited to $f_{\max}$. The Fourier transform is non-zero only at intervals of $f_P = 1/P$. In every exposure duration observe different linear

combinations of the periodic signal. To solve this problem, we simply use basis pursuit de-noising, described above.

Other applications include of compressive sensing include iris recognition. The objective is to recognize a person from the texture features on his/hers iris image. Iris images acquired from a partially cooperating user often suffer from specular reflections, segmentation error, occlusion, and blur. We formulate this problem as follows. Assume L classes and n images per class in a gallery $D_k = [x_1, ..., x_k]$. The training images of the kth class is represented as

$$\mathbf{D} = [D_1, ..., D_k] \in \mathbb{R}^{N \times (n,L)} = [x_{11}, ..., x_{1n} | x_{21}, ..., x_{2n} | ... | x_{L1}, ..., x_{Ln}] \tag{405}$$

The dictionary D is obtained by concatenating all the training images. The unknown test vector can be represented as a linear combination of the training images as

$$y = \sum_{i=1}^{L} \sum_{j=1}^{n} \alpha_{ij} \mathbf{x}_{ij} \tag{406}$$

Only the α that is provided is non-zero for the corresponding x_{ij}. As a result α is a very sparse vector, which is where compressive sensing methodology can be used. We make the assumption that the test image can be written as a linear combination of the training images of the correct class alone. So the coefficient α is a sparse vector. Hence α can be recovered by basis pursuit:

$$\hat{\alpha} = \arg \min_{\alpha' \in \mathbb{R}^N} ||\alpha||_1 \quad \text{subject to} \quad y = \mathbf{D}\alpha' \tag{407}$$

When the test image is well acquired, the coefficient vector α will be sparse. A measure of sparsity is the Sparse Concentration Index SCI, defined by

$$SCI(\alpha) = \frac{\frac{L \cdot \max ||\prod_i(\alpha)||}{||\alpha||_1} - 1}{L - 1} \tag{408}$$

SCI measures the fraction of the energy present in the "best" class. So well acquired images will have high SCI. Hence, we reject images with a low SCI. We have effectively discussed a selection and recognition algorithm, summarized below.

(1) Given the gallery, construct the dictionary D by arranging the training images as its columns.
(2) Using the test image, by Basis Pursuit, obtain the coefficient vector α.
(3) Obtain the Sparsity Concentration Index (SCI).
(4) Compare SCI with a threshold to reject the poorly acquired images.
(5) Find the reconstruction error while representing the test image with coefficients of each class separately.
(6) Select the class giving the minimum reconstruction error.

For this problem, we have a need a for cancelability. In other words, we apply a revocable and non invertible transformation on the original one. Random Projections (RP) can be utilized due to Johnson Lindenstrauss (JL) lemma. One such mapping is projection using a Random Matrix. Let g be the N dimensional Gabor features of the image. g is projected randomly onto a subspace of dimension n as follows

$$y = \Phi g \in \mathbb{R}^n \tag{409}$$

Recognition is performed on the vector y. Face recognition is also possible via sparse representations (Wright et. al PAMI 2009). This paper employed a similar methodology similar to what was described above for iris identification. Once α is recovered, then we choose the labels closest to y. Fro all the methodology described above, the dictionary D was assumed to be known. However, it is possible to learn the dictionaries. Many such algorithms exist for dictionary learning. One famous example is KSVD. Another is the method of optimal directions.

K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation

Given a set of training examples $\mathbf{B} = [x_1, ..., x_n]$, we want to find the dictionary that leads the best representation for each member in this set, under strict sparsity conditions. To do so, The KSVD algorithm is similar to the K means algorithm. For KSVD, our optimization is

$$\arg\min_{D,\Gamma} ||B - D\Gamma||_F^2 \quad \text{subject to } ||\gamma_i||_0 \leq T_0, \forall\ i \tag{410}$$

We input our initial dictionary $D^{(0)} \in \mathbb{R}^{N \times P}$, with normalized columns and signal matrix $B = [x_1, ..., x_n]$ and sparsity level T_0. The output of this algorithm is a trained dictionary D with representation matrix Γ. The procedure is as follows.

(1) Set $J = 1$. Repeat until convergence
(2) Sparse coding stage: Use any pursuit algorithm to compute the sparse representation vectors γ_i for each signal $[x_1, ...x_m]$
(3) Dictionary update stage: for each column $k = 1, ..., P \in D^{J-1}$ update by:
  (a) Define the group of examples that use $\omega_k = \{i | 1 \leq i \leq P, \gamma_T^k(i) \neq 0\}$
  (b) Computer the overall representation error matrix E_k:

$$E_k = B - \sum_{j \neq k} d_j \gamma_T^j \tag{411}$$

  (c) Restrict E_k by choosing columns that correspond to ω_k and obtain E_k^R
  (d) Apply SVD decomposition $E_k^R = U \Delta V^T$. Select the update dictionary column to the first column of U. Update the coefficient vector to be the first column of V multiplied by $\Delta(1, 1)$
(4) Set $J = J + 1$

Effectively, we fix D and learn γ by basis pursuit. Then we can improve D based on this new γ. We can iteratively do this procedure until convergence.

Now we formalize the mathematics of K-SVD. The sparse representation solution of either

$$(P_0) \quad \min_x ||x||_0 \quad \text{subject to} \quad y = Dx \tag{412}$$

or

$$(P_{0,\epsilon}) \quad \min_x ||x||_0 \quad \text{subject to} \quad ||y - Dx|| \leq \epsilon \tag{413}$$

where the former is known as basis pursuit and latter, Basis Pursuit De-Noising. The next step involves updating D. One option is the maximum likelihood estimator. In our model we have $y = Dx + v$, where x is the sparse representation and v is a Gaussian white noise residual vector with variance σ^2. If we consider the likelihood $P(Y|D)$, assuming iid likelihoods for each y_i,

$$P(Y|D) = \prod_{i=1}^{N} P(y_i|D) \tag{414}$$

In terms of a hiddle variable x, we have

$$P(y_i|D) = \int P(y_i, x|D)dx = \int P(y_i|x, D) \cdot P(x)dx \tag{415}$$

With our noise approximation, we have

$$P(y_i|x, D) \propto \exp\left\{\frac{1}{2\sigma^2}||Dx - y_i||^2\right\} \tag{416}$$

Assuming that the prior $P(x)$ takes the form of a Laplace distribution

$$P(x) = \exp\{\lambda||x||_1\} \tag{417}$$

Our likelihood is now

$$P(y_i|D) = \int P(y_i, x|D)dx = \int P(y_i|x, D) \cdot P(x)dx \propto \int \exp\left\{\frac{1}{2\sigma^2}||Dx - y_i||^2\right\} \cdot \exp\{\lambda||x||_1\}dx \tag{418}$$

The ML estimator can be shown to be

$$D = \arg\max_D \sum_{i=1}^{N} \max_{x_i}\{P(y_i, x_i|D)\} = \arg\min_D \sum_{i=1}^{N} \min_{x_i}\left\{||Dx_i = y_i||^2 + \lambda||x_i||_1\right\} \tag{419}$$

We further constrain the problem by the $l2$ norm to ensure convergence, for which we have the following update rule for the dictionary:

$$D^{(n+1)} = D^{(n)} - \eta \sum_{i=1}^{N} (D^{(n)}x_i - y_i)x_i^T \tag{420}$$

Alternatively, to learn the dictionary, we can use a maximum a-posteriori (MAP) approach. From Bayes rule, we know that the posterior $P(D|Y) \propto P(Y|D)P(D)$. In the case where we constrain the prior to have Euclidean norm 1, then the update rule is:

$$D^{(n+1)} = D^{(n)} + \eta EX^T + \eta \cdot \text{tr}(XE^T D^{(n)})D^{(n)} \tag{421}$$

where E is the mean square error. We can alternatively constrain the dictionary to have columns with unit l_2 norm. In this case, the update rule for each colum is:

$$d_i^{(n+1)} = d_i^{(n)} + \eta\big(I - d_i^{(n)}d_i^{(n)^T}\big)E \cdot x_i^T \tag{422}$$

One very appealing algorithm is MOD (method of optimal directions) for dictionary training. MOD assumes that at any particular stage we have x from basis pursuit. We then construct a sum of squares error, minimize by differentiating, and defining an update rule. Therefore, by performing basis pursuit iteratively, we can obtain the new dictionary. Mathematically, let $e_i y_i - Dx_i$. The overall mean square error is

$$||E||_F^2 = ||[e_1, e_2, ..., e_N]||_F^2 = ||Y - DX||_F^2 \tag{423}$$

Here $||X||_F$ is the Euclidean norm, and the matrices X and Y correspond to concatenated data. We minimize the error by

$$\frac{\partial||E||_F^2}{\partial D} = (Y - DX)X^T = 0 \implies D^{(n+1)} = YX^{(n)^T} \cdot (X^{(n)}X^{(n)^T})^{-}1 \tag{424}$$

It can be shown that MOD and the ML method result in identical update equations. Of all algorithms for dictionary update, KSVD has persisted, which is a generalization of K means. For this algorithm, we have to define how sparse we will. For some sparsity, we represent the data samples by nearest neighbor by solving

$$\min_{D,X}\{||Y - DX||_F^2\} \quad \text{subject to} \quad \forall\, i, ||x_i||_0 \leq T_0 \tag{425}$$

The K-SVD algorithm is as follows. We initialize the dictionary matrix with $D^{(0)} \in \mathbb{R}^{n\times K}$ with l^2 normalized columns. We have two steps.

(1) Sparse Coding Stage. The sparse coding sage involves using Basis pursuit or another pursuit algorithm to approximate x using the dictionary from the previous iteration:

$$\forall i, \quad \min_{x_i} ||y_i - Dx_i||_2^2 \quad \text{subject to} \quad ||x_i||_0 \leq T_0 \tag{426}$$

(2) Codebook Update Stage. For each column $k \in D^{J-1}$, we update by defining a group of examples $\omega_k = \{i|1 \leq i \leq N, x_T^k i \neq 0\}$. Compute the overall representation matrix $E_k = Y - \sum_{j\neq k} d_j x_T^j$. Restrict columns of E_k to ω_k to obtain E_k^R. Apply SVD decomposition $E_k^R = U\Delta V^T$, and choose the update dictionary columns to be the first column of U and the updated coefficient vector to be the first column of V multiplied by $\Delta(1, 1)$.

Mathematically, we represent the K-SVD algorithm as follows. Recall that our desired optimization is:

$$\min_{D,X}\{||Y - DX||_F^2\} \quad \text{subject to} \quad \forall\, i, ||x_i||_0 \leq T_0 \tag{427}$$

For the sparse coding stage, we assume that D is fixed. As a result, we have

$$||Y - DX||_F^2 = \sum_{i=1}^{N} ||y_i - Dx_i||_2^2 \tag{428}$$

As a result, our objective takes the form

$$\min_{x_i}\{||y_i - Dx_i||_2^2\} \quad \text{subject to} \quad \forall\, i, ||x_i||_0 \leq T_0 \tag{429}$$

This solution is good for small T_0 and can be determined via basis pursuit. Now, we consider the codebook update stage. Here, we assume that all X and D are fixed except for column d_k and corresponding coefficients x_T^k. Here, the objective function is:

$$\begin{aligned} ||Y - DX||_F^2 &= \left|\left|Y - \sum_{j=1}^{K} d_j x_T^j\right|\right|_F^2 = \left|\left|\left(Y - \sum_{j\neq k} d_j x_T^j\right) - d_k x_T^k\right|\right|_F^2 \\ &= ||E_k - d_k x_T^k||_F^2 \end{aligned} \tag{430}$$

In order to ensure sparsity, we only consider components of non-zero $x_T^l(i)$:

$$\omega_k = \{i | 1 \leq i \leq N, x_T^k i \neq 0\} \tag{431}$$

Next we define a matrix Ω_k of size $N \times |\omega_k|$ with ones in $(\omega_k(i), i)$ and zeros elsewhere. Let $x_R^k = \Omega_k x_T^k$ (i.e. vector of only non-zero x_T^k). Thus our minimization simplifies to:

$$||E_k\Omega_k - d_k x_T^k \Omega_k||_F^2 = ||E_k^R - d_k x_R^k||_F^2 \tag{432}$$

on which SVD can be performed, as described in the algorithm above.

Next let us consider compressive sensing for a video. Here, a video is modeled as a first order Gaussian Markov process:

$$y(t) = Cx(t) + w(t) \tag{433}$$

$$x(t+1) = Ax(t) + v(t) \tag{434}$$

where $x(t)$ is the hidden state vector, A is the transition matrix, and C is the observation matrix. Rather than working with y, is is possible to work with $y' = \Phi y$, to allow for video reconstruction. Furthermore, the K-SVD can be applied to non-linear models of data, in a similar method to what is described above.

Cameras and Calibration

Prior to a detailed discussion of cameras and calibration, we will first provide a brief introduction to some basic mathematical concepts, namely line segments and rays, homogeneous coordinates, and 2D and 3D transformations. A segment between two points can be constructed as follows. A point r will be in the segment between p and q if there exists a λ such that

$$r = (1 - \lambda)p + \lambda q, \quad 0 \leq \lambda \leq 1 \tag{435}$$

A point q falls on a ray starting at p in the direction v if

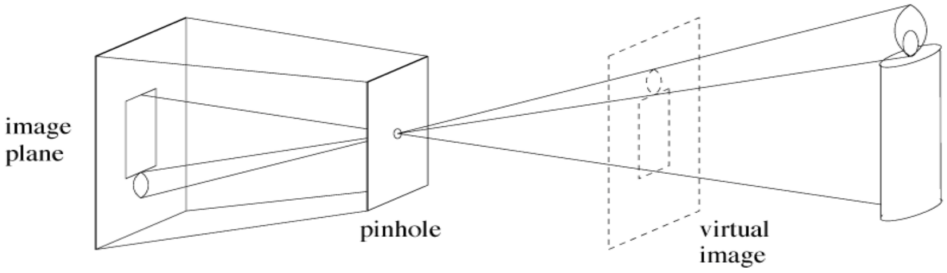
$$q = p + \lambda v, \quad \lambda \geq 0 \tag{436}$$

Next we will define the notion of homogeneous coordinates. First we define a binary relation $\sim$ where $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$ if a λ exists such that $(x_1, y_1, z_1) = (x_2, y_2, z_2)$. $\sim$ is an equivalence relation (i.e. this relation is reflexive, symmetric, and transitive). Homogeneous coordinates have the following properties:

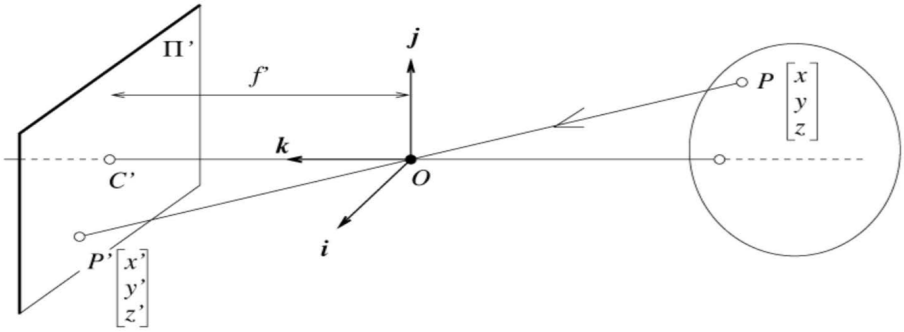
(1) Any point on the plane is represented by (x, y, z) where x,y and z are not all 0
(2) The point is unchanged if the coordinates are multiplied by a common factor.
(3) When z is not 0 (x, y, z) represents $(x/z, y/z)$ in the euclidean plane.
(4) When z is 0 the point is represented is the point at infinity.

We will use $\tilde{x}$ to represent homogeneous coordinates. Next, we look into some simple transformations. 2D transformations include translation, rotation and translation, scaled rotation and translation (similarity), affine, and projective transforms. 3D transformations include similarity, translation, RT, affine, and projective transforms. Transformations are encoded by a matrix that operates on points and involve matrix multiplications. We will now study 3D to 2D projections, known as perspective.

We begin by introducing the idea of camera obscura, which involves the idea of a large room with a pinhole, through while light enters for image generation. In its modern form, cmaera obscuras are known as pinhole cameras. Consider an abstract camera model as a box with a small hole in it. Pinhole camers work in practice. An example of such a camera is shown below:



Next we will derive the equation of projection, shown below:

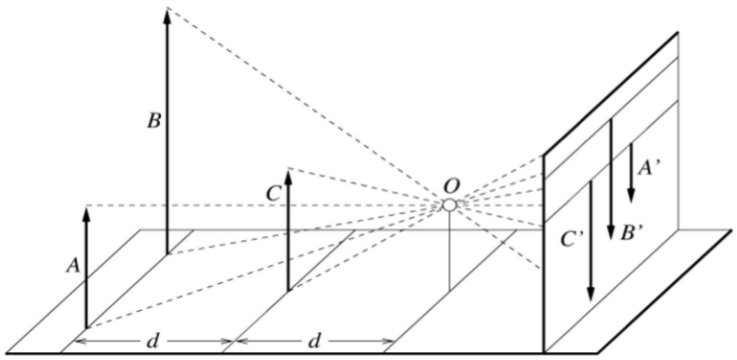


In Cartesian coordinates, we have, by similar triangles, that $(x, y, z) \rightarrow (fx/z, fy/z, f)$. We ignore the third coordinate and obtain

$$(x, y, z) \rightarrow \left(f\frac{x}{z}, f\frac{y}{z}\right) \tag{437}$$

Next we consider perspective projection. A camera is located at $(0, 0, 0)$ looking in the positive z direction. A point in the world (x, y, z) will appear on an image plane at $z = f$ at the location $(fx/z, fy/z, f)$. In other words, we shoot a ray from the camera location to the world and see where it intersects the image plane. For instance, $(p_x, p_y, p_z) = (0, 0, 0) + \lambda(x, y, z)$, we say that the image plane $p_z = f$. On the image plane $(p_x, p_y, f) = (\lambda x, \lambda y, \lambda z)$ so $\lambda = f/z$ and $p_x = fx/z$, $p_y = f\lambda/z$. In summary, on a camera at the origin looking in the z direction, a world point (x, y, z) projects to $fx/z, fy/z, f$. Next, we consider two examples: (1) Distance objects are smaller, and (2) Parallel lines meet at the horizon.

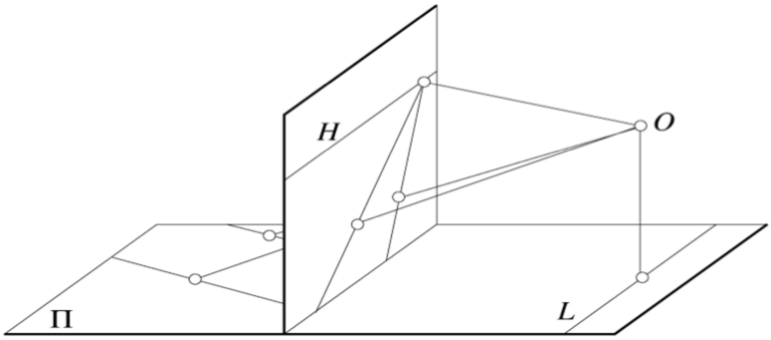
Distance Objects are Smaller



Say we have a camera at the origin with focal length 1. Let $p_1 = (x, 0, z)$ and $p_2 = (x, y, z)$ (i.e. 3D distance between both points is y). Additionally, $p_3 = (x, 0, 2z)$ and $p_4 = (x, y, 2z)$ (i.e. 3D distance between both points is also y). However, p_1 and p_2 are closer to the camera than p_3 and p_4. Now let us do perspective projection. Here $t(p_1) = (fx/z, 0)$ and $t(p_2) = (fx/z, fy/z)$, and the 2D distance between both points if fy/z. Similarly, $t(p_3) = (fx/2z, 0)$ and $t(p_4) = (fx/2z, fy/2z)$, and the 2D distance is $fy/2z$.

Parallel Lines Meet

It is common to draw image planes in front of the focal point. Moving the image plane merely scales the image.



For this case, the camera is at the origin, looking in the positive z direction. Assume that $y = -1$ is a ground plane. Assume two distinct parallel lines on the ground plane, namely $x_1 = az_1 + b_1$ and $x_2 = az_2 + b_2$. $(x_1, -1, z_1)$ is a point on line 1, and $(x_2, -1, z_2)$ is a point on line 2. Applying perspective projection yields: $(x_1/z_1, -1/z_1) = ((az_1 + b)/z_1, -1/z_1) = (a, 0)$ as $z \to \infty$ and $(x_2/z_2, -1/z_2) = ((az_2 + b)/z_2, -1/z_2) = (a, 0)$ as $z \to \infty$. Thus, parallel lines meet at the horizon, and the point depends on the slope. All of this is independent of the intercepts b_1 and b_2.

Next consider the example of front to parallel stereo. We have 2 cameras, one located at the origin $(0, 0, 0)$ and another located at $(T, 0, 0)$, looking in the positive z direction at a point at (x, y, z). On the first camera, this point is located at $(fx/z, fy/z, f)$. To determine the point on the second camera, we need to translate the points $(x, y, z) - (T, 0, 0)$. After perspective projection, we have $(f(x - T)/z, fy/z, f)$. The disparity, d is the 2D difference between these points and is $d = fT/z \implies z = fT/d$.

The equations of weak perspective simplify the problem to $(x, y, z) \to s(x, y)$. s is constant for all points. In this case, parallel lines no longer converge. The weak perspective involves much simpler math, is accurate when the object is small and distance and is most useful for recognition. However, the pinhole perspective is much more accurate for scenes (used in structure from motion). When accuracy really matters, we must model real cameras. In summary, cameras map the 3d world to 2d images. Understanding how cameras work allows us to reason about the the 3d world using 2d images.

Camera Calibration

Formally, the perspective equations are:

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} = \frac{f}{x_3} \begin{pmatrix} x_1 \\ x_2 \\ \frac{x_3}{f} \end{pmatrix} \sim \begin{pmatrix} x_1 \\ x_2 \\ \frac{x_3}{f} \end{pmatrix} \tag{438}$$

In linear algebraic terms

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f} & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} \implies \mathbf{y} \sim \mathbf{C}\mathbf{x} \tag{439}$$

The above matrix is called the camera matrix C. We have studied interesting properties of C. This C is still the ideal case. Now, let us consider the location and orientation of the camera. These are two 4×4 matrices that can be composed, one for rotation and one for translation:

$$\begin{pmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \tag{440}$$

The composition is just the multiplication of the two matrices:

$$\mathbf{x} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \mathbf{x}' \tag{441}$$

These are called the extrinsic parameters. We indirectly used this when we did the stereo example: we applied a translation, to the points in the world. 3D rotations align the world coordinates with camera coordinates. The real form of the camera matrix (K) in the real world

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{442}$$

This matrix describes how 3D points in the world generate pixels in the image. The parameters in K are called the intrinsic parameters. Putting this all together

$$z_c = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} = M \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{443}$$

where $M = K[R \quad T]$. Finding the parameters is commonly referred to as calibration. Direct linear transformation and Zhang are the most common methods. Solve the parameters by matching the images acquired by a given camera to the expected known, scene. The techniques are based on numeric optimization. We will end with the idea of a Direct Linear Transformation. Conceptually, we have many 2D points (x_k) and the corresponding 3D points (y_k).

$$\mathbf{x}_k = \mathbf{A}\mathbf{y}_k \quad k = 1, ..., N \tag{444}$$

We write the variables in A (which in this case is K) as a system of equations. Find a least squares solution for the variables in A. In direct linear transformation we have:

$$\mathbf{x}_k \propto \mathbf{A}\mathbf{y}_k \quad k = 1, ..., N \tag{445}$$

where the values may be related by a multiplicative factor (remember $\sim$), that is unknown. This is a slight subtlety the technique is the same.

Computation of Optical Flow

Let $E(x, y, t)$ be the intensity over space and time. Over some time δt, our new intensity is $E(x+\delta x, y+\delta y, t+\delta t)$. We make the assumption of brightness constancy, which states that as the pixel moves, the intensity stays constant. As a result,

$$E(x, y, t) = (x + \delta x, y + \delta y, t + \delta t) \tag{446}$$

We can expand the RHS of this equation about $E(x, y, t)$,

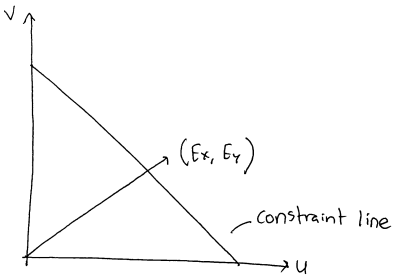
$$E(x, y, t) = E(x, y, t) + \delta x \frac{\partial E}{\partial x} + \delta y \frac{\partial E}{\partial y} + \delta t \frac{\partial E}{\partial t} \tag{447}$$

Dividing through by δt and taking the limit as $\delta t \to 0$:

$$\begin{aligned} \frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} &= 0 \\ E_x u + E_y v + E_t &= 0 \end{aligned} \tag{448}$$

where $x = dx/dt$ and $v = dy/dt$. This expression is known as the optical flow constraint equation. This is very similar to the continuity equation from fluid mechanics. At each pixel, we have u and v, which means we cannot get these velocity from one frame. Alternatively, we can write, in matrix form, the optical flow constraint equation as

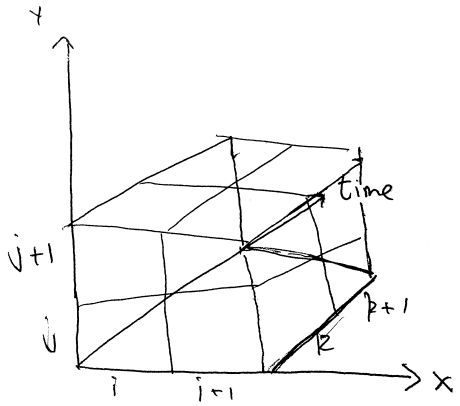
$$\begin{pmatrix} E_x & E_y \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} = -E_t \tag{449}$$



The component of the motion in the direction of the brightness gradient (E_x, E_y) is simply $-E_t/\sqrt{E_x^2 + E_y^2}$. In the (u, v) coordinate system, exists on a constraint line that is perpendicular to the brightness gradient, as shown in the figure on the left, with distance from the origin given by $-E_t/\sqrt{E_x^2 + E_y^2}$. To solve for both u and v, we solve the so called "aperture problem." One such consideration is the barber pole illusion, in which a rotating object appears to be translating in space. To solve this problem, we need to introduce additional constraints. In order to evaluate the continuity equations above, we require an approximation for continuous derivatives, as follows:

$$E_x = \frac{1}{4\delta x}[(E_{i+1,j,k} + E_{i+1,j,k+1} + E_{i+1,j+1,k} + E_{i+1,j+1,k+1}) - (E_{i,j,k} + E_{i,j,k+1} + E_{i,j+1,k} + E_{i,j+1,k+1})] \tag{450}$$

Geometrically, this difference can be viewed as a local average, as shown below.



In order to determine u and v, we need additional constraint. Define the error constraint e_c as:

$$e_c = \iint_{\text{image}} (E_x u + E_y v + E_t)^2 dxdy \tag{451}$$

We desire e_c to be minimized. We also want to add a smoothness constraint, e_s:

$$e_s = \iint_{\text{image}} [(u_x^2 + u_y^2) + (v_x^2 + v_y^2)] dxdy \tag{452}$$

The problem now involves finding (u, v) such that the total error $e_s + \lambda e_c$ is minimized, where λ is the weighting factor. This is an example of the so called regularization approach. To solve for u and v, we can either: (1) solve by discretization and (2) calculus of variations. Let us first consider solving this system discretely. Consider an image pixel (i, j). We can determine the departure from the smoothness constraint as

$$s_{ij} = \frac{1}{4}[(u_{i+1,j} - u_{ij})^2 + (u_{i,j+1} - u_{ij})^2 + (v_{i+1,j} - v_{ij})^2 + (v_{i,j+1} - v_{ij})^2] \tag{453}$$

Additionally, the error in the optical flow continuity constraint equation is:

$$c_{ij} = (E_x^{ij} u_{ij} + E_y^{ij} v_{ij} + E_t^{ij})^2 \tag{454}$$

The total error is thus

$$e = \sum_i \sum_j (s_{ij} + \lambda c_{ij}) \tag{455}$$

Next, we differentiate the total error e with respect to u_{kl} and v_{kl} and set equal to zero to determine the optimal u and v. Let $\bar{u}_{kl}$ and $\bar{v}_{kl}$ are the local temporal averages (e.g. 4 or 8 neighborhood averages for an image). To find the derivatives for s_{ij} with respect to u_{kl} and v_{kl}, we must consider only indices in which u_{kl} and v_{kl} appear, namely $i = k, j = l$, $i = k - 1, j = l$, and $i = k, j = l - 1$. Consider these indices and simplifying leads to the following expression:

$$\frac{\partial e}{\partial u_{kl}} = 2(u_{kl} - \bar{u}_{kl}) + 2\lambda(E_x^{kl} u_{kl} + E_y^{kl} v_{kl} + E_t^{kl}) E_x^{kl} = 0 \tag{456}$$

$$\frac{\partial e}{\partial v_{kl}} = 2(v_{kl} - \bar{v}_{kl}) + 2\lambda(E_x^{kl} u_{kl} + E_y^{kl} v_{kl} + E_t^{kl}) E_y^{kl} = 0 \tag{457}$$

These equations are coupled. We can determine an iterative recursion for u_{kl} and v_{kl} as follows:

$$u_{kl}^{n+1} = \bar{u}_{kl}^n - \frac{E_x^{kl} \bar{u}_{kl}^n + E_y^{kl} \bar{v}_{kl}^n + E_t^{kl}}{1 + \lambda[(E_x^{kl})^2 + (E_y^{kl})^2]} E_x^{kl} \tag{458}$$

$$v_{kl}^{n+1} = \bar{v}_{kl}^n - \frac{E_x^{kl} \bar{u}_{kl}^n + E_y^{kl} \bar{v}_{kl}^n + E_t^{kl}}{1 + \lambda[(E_x^{kl})^2 + (E_y^{kl})^2]} E_y^{kl} \tag{459}$$

Convergence requires a large regularization parameter λ, resulting in very smooth solutions. Next, we consider the solution using calculus of variations. Recall, from above, we desire a minimum for

$$e = \sum_i \sum_j (s_{ij} + \lambda c_{ij}) \tag{460}$$

where

$$s_{ij} = \frac{1}{4}[(u_{i+1,j} - u_{ij})^2 + (u_{i,j+1} - u_{ij})^2 + (v_{i+1,j} - v_{ij})^2 + (v_{i,j+1} - v_{ij})^2] \tag{461}$$

$$c_{ij} = (E_x^{ij} u_{ij} + E_y^{ij} v_{ij} + E_t^{ij})^2 \tag{462}$$

The Euler-Lagrange equations for u and v his optimization are:

$$F_u - \frac{\partial}{\partial x}F_{u_x} - \frac{\partial}{\partial y}F_{u_y} = 0 \tag{463}$$

$$F_v - \frac{\partial}{\partial x}F_{v_x} - \frac{\partial}{\partial y}F_{v_y} = 0 \tag{464}$$

where

$$F = \alpha^2[(u_x^2 + u_y^2) + (u_x^2 + u_y^2)] + (E_x u + E_y v + E_t)^2 \tag{465}$$

Evaluating the differentials for the Euler-Lagrange equations described above:

$$F_u = 2(E_x u + E_y v + E_t)E_x \tag{466}$$

$$F_{u_x} = 2\alpha^2 u_x \implies \frac{\partial}{\partial x}F_{u_x} = 2\alpha^2\frac{\partial^2 u}{\partial x^2} \tag{467}$$

$$F_{u_y} = 2\alpha^2 u_y \implies \frac{\partial}{\partial y}F_{u_y} = 2\alpha^2\frac{\partial^2 u}{\partial y^2} \tag{468}$$

From our Euler Lagrange equations:

$$\begin{aligned} F_u &= \frac{\partial}{\partial x}F_{u_x} + \frac{\partial}{\partial y}F_{u_y} \\ &\implies 2(E_x u + E_y v + E_t)E_x = 2\alpha^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 2\alpha^2\nabla^2 u \\ &\implies \alpha^2\nabla^2 u - E_x E_t = E_x^2 u + E_x E_y v \end{aligned} \tag{469}$$

Likewise, by symmetry, our Euler-Lagrange equation for v is:

$$\alpha^2\nabla^2 v - E_y E_t = E_x E_y u + E_y^2 v \tag{470}$$

To solve these equations, we first make the following approximations for the Laplacian:

$$\nabla^2 u = k(\bar{u} - u) \tag{471}$$

$$\nabla^2 v = k(\bar{v} - v) \tag{472}$$

Substituting these into our Euler-Lagrange equations yields:

$$\alpha'^2(\bar{u} - u) - E_x E_t = E_x^2 u + E_x E_y v \tag{473}$$

$$\alpha'^2(\bar{v} - v) - E_y E_t = E_x E_y u + E_y^2 v \tag{474}$$

where $\alpha'^2 = k\alpha^2$. Solving this system of equations yields the following recursions:

$$u^{n+1} = -\frac{E_x(E_x\bar{u}^n + E_y\bar{v}^n + E_t)}{(\alpha'^2 + E_x^2 + E_y^2)} \tag{475}$$

$$v^{n+1} = -\frac{E_y(E_x\bar{u}^n + E_y\bar{v}^n + E_t)}{(\alpha'^2 + E_x^2 + E_y^2)} \tag{476}$$

In low texture regions, gradients have small magnitude, thus preventing application of optical flow in these situations. An alternative approach is known as the Lucas-Kanade approach for optical flow. Recall that for a single velocity for all pixels within an image patch,

$$E(u, v) = \sum_{x,y\in\Omega}(I_x(x, y)u + I_y(x, y)v + I_t)^2 \tag{477}$$

with corresponding derivatives

$$\frac{dE(u,v)}{du} = \sum 2I_x(I_x u + I_y v) \tag{478}$$

$$\frac{dE(u,v)}{dv} = \sum 2I_y(I_x u + I_y v) \tag{479}$$

We can write the solution with the following matrix equation:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = - \begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix} \tag{480}$$

This can be written as the sum of a 2×2 outer product
0

$$\left(\sum \nabla I \nabla I^T \right) \vec{U} = - \sum \nabla I I_t \tag{481}$$

This method was particularly effective as there are no iterations, compared to other methods. Yet another method is the course to fine estimation, which states that, for small u and v,

$$I_x \cdot u + I_y \cdot v + I_t \approx 0 \tag{482}$$

Nagel proposed the following optical flow algorithm. The problem is formulated as the minimzation of the function

$$\iint (\nabla I^T \mathbf{v} + I_t)^2 + \frac{\alpha^2}{||\nabla I||_2^2 + 2\delta} \times [(u_x I_y - u_y I_x)^2 + (v_x I_y - v_y I_x)^2] + \delta(u_x^2 + u_y^2 + v_x^2 + v_y^2) dx dy \tag{483}$$

The soolution to this is

$$u^{k+1} = \xi(u^k) - \frac{I_x(I_x \xi(u^k) + I_y \xi(v^k) + I_t)}{I_x^2 + I_y^2 + \alpha^2} \tag{484}$$

$$v^{k+1} = \xi(u^k) - \frac{I_y(I_x \xi(u^k) + I_y \xi(v^k) + I_t)}{I_x^2 + I_y^2 + \alpha^2} \tag{485}$$

$$\tag{486}$$

where

$$\xi(u^k) = \bar{u}^k - 2I_x I_y u_{xy} - q^T(\nabla u^k) \tag{487}$$

$$\xi(v^k) = \bar{u}^k - 2I_x I_y v_{xy} - q^T(\nabla v^k) \tag{488}$$

$$q = \frac{1}{I_x^2 + I_y^2 + 2\delta} \nabla I^T \left[\begin{bmatrix} I_{yy} & -I_{xy} \\ -I_{xy} & I_{xx} \end{bmatrix} + 2 \begin{bmatrix} I_{yy} & I_{xy} \\ I_{xy} & I_{xx} \end{bmatrix} W \right] \tag{489}$$

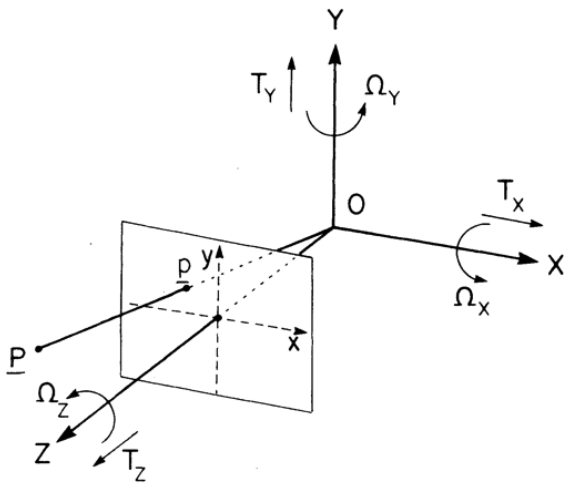
The sum of square differences method involves

$$SSD_{1,2}(\mathbf{x}; \mathbf{d}) = \sum_{j=-n}^{n} \sum_{i=-n}^{n} W(i,j) \times [I_1(\mathbf{x} + (i,j)) - I_2(\mathbf{x} + \mathbf{d} + (i,j))]^2 = W(x) * [I_1(\mathbf{x} + \mathbf{d})]^2 \tag{490}$$

where W is a window function. Here we find the $\mathbf{d}$ that minimizes the error. To assess these algorithms, images with known velocities were employed. Accuracy was assessed by determining: (1) the optical flow density (number of pixels with optical flow), and (2) accuracy (dot product).

Optical Flow, Motion, and Structure

In this section, optical flow will be related to the motion and structure of a moving surface. Consider a point P with coordinates (X, Y, Z) and its position on the image plane $p = (x, y)$. Furthermore, we assume a focal length $f = 1$. The point P is moving some with some angular velocity $(\Omega_x, \Omega_y, \Omega_z)$ and translational velocity (T_x, T_y, T_z). This is schematically depicted below.



Here, we have a viewer centered coordinated system. Let translational velocity of the camera be

$$\mathbf{T} = \begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} \tag{491}$$

Furthermore, let the rotational component be

$$\mathbf{\Omega} = \begin{pmatrix} \Omega_X \\ \Omega_Y \\ \Omega_Z \end{pmatrix} \tag{492}$$

The velocity of the particle P is:

$$\frac{dP}{dt} = \mathbf{T} + \mathbf{\Omega} \times P \tag{493}$$

where $\times$ denotes the cross product between two vectors:

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= (a_1 i + a_2 j + a_3 k) \times (b_1 i + b_2 j + b_3 k) \\ &= (a_2 b_3 - a_3 b_2) i + (a_3 b_1 - a_1 b_3) j + (a_1 b_2 - a_2 b_1) k \end{aligned} \tag{494}$$

Thus

$$\frac{dP}{dt} = \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = \begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} + \begin{pmatrix} \Omega_Y Z - \Omega_Z Y \\ \Omega_Z X - \Omega_X Z \\ \Omega_X Y - \Omega_Y Z \end{pmatrix} \tag{495}$$

The central (or perspective) projection involves writing the image coordinates (x, y) in terms of the spatial coordinates (X, Y, Z). Recall that from the perspective equations, we have

$$x = \frac{X}{Z} \implies \dot{x} = \frac{\dot{X}}{Z} - \frac{X\dot{Z}}{Z^2} \tag{496}$$

Let $\dot{x} = u$. Then substituting in our equations for X and Z as well as their derivatives, we have

$$\begin{aligned}\dot{x} = u &= \frac{1}{Z}(T_X + \Omega_Y Z - \Omega_Z Y) - \frac{X}{Z^2}(T_Z + \Omega_X Y - \Omega_Y X) \\ &= \frac{T_X}{Z} + \Omega_Y - \Omega_Z \frac{Y}{Z} - \frac{X T_Z}{Z^2} - \frac{X}{Z}\frac{Y}{Z}\Omega_X + \Omega_y \frac{X^2}{Z^2}\end{aligned} \tag{497}$$

Substituting in (x, y) image plane coordinates from the perspective equations yields

$$\begin{aligned}u = \frac{T_X}{Z} + \Omega_Y - \Omega_Z y &= \frac{T_Z x}{Z} - xy\Omega_X + \Omega_Y x^2 \\ &= \Omega_X xy + \Omega_Y(1 + x^2) - \Omega_Z y + \frac{T_X - xT_Z}{Z} = u_R + u_T\end{aligned} \tag{498}$$

where u_R is the rotational component and u_T is the translational component. Note that the rotational component is independent of Z and thus independent of structure. Structural information is tied only to the translational component. We can do something similar for v. To be consistent with Adiv's notations, let $u = \alpha$ and $v = \beta$. From above, we know that $\alpha = \alpha_T + \alpha_R$, where

$$\alpha_T = \frac{T_X - xT_Z}{Z} \tag{499}$$

$$\alpha_R = -\Omega_X xy + \Omega_Y - \Omega_Z y + \Omega_Y x^2 \tag{500}$$

are the translational and rotational components, respectively. To get β, we repeat the same process. Recall that $y = Y/Z$ and $\beta = \dot{y}$.

$$\begin{aligned}\beta = v &= \frac{\dot{Y}}{Z} - \frac{Y\dot{Z}}{Z^2} \\ &= -\Omega_X(1 + y^2) + \Omega_Y xy + \Omega_Z x + \frac{T_y - T_Z y}{Z} = \beta_T + \beta_R\end{aligned} \tag{501}$$

The translational and rotational components for β are thus:

$$\beta_T = \frac{T_Y - T_Z y}{Z} \tag{502}$$

$$\beta_R = -\Omega_X(1 + y^2) + \Omega_Y xy + \Omega_Z x \tag{503}$$

To solve for structure from motion, we write α, β in terms of $\mathbf{\Omega}$ and $\mathbf{T}$. These equations can be solved with many assumptions, including the planar patch assumption, in which rotations were neglected. Let $f_1(\mathbf{\Omega}, \mathbf{T}, Z)$ and $f_2(\mathbf{\Omega}, \mathbf{T}, Z)$ be two functions that represent the structure of the problem for α and β, respectively. Over sum number of pixels, we can estimate structure information as follows:

$$\min_{\mathbf{\Omega}, \mathbf{T}, Z} \sum_i (\alpha_i - f_1(\mathbf{\Omega}, \mathbf{T}, Z))^2 + \sum_i (\beta_i - f_2(\mathbf{\Omega}, \mathbf{T}, Z))^2 \tag{504}$$

$f_1(\mathbf{\Omega}, \mathbf{T}, Z)$ and $f_2(\mathbf{\Omega}, \mathbf{T}, Z)$ depend on the geometrical assumptions for the objects. Thus, we can determine structure from optical flow. Next we consider optical flow for a 3D planar patch.

Optical Flow for a 3D Planar Patch

For the case of a planar patch, we have the constraint

$$k_x X + k_y Y + k_z Z = 1 \tag{505}$$

Recall from the perspective equations that $x = X/Z$ and $y = Y/Z$, which implies that $X = xZ$ and $Y = yZ$. Substituting these into our plane yields

$$\begin{aligned} k_x(xZ) + k_y(yZ) + k_z Z &= Z(k_x x + k_y y + k_z) = 1 \\ \implies \frac{1}{Z} &= k_x x + k_y y + k_z \end{aligned} \tag{506}$$

Recall that

$$\begin{aligned} \alpha &= \Omega_X xy + \Omega_Y(1+x^2) - \Omega_Z y + \frac{T_X - xT_Z}{Z} \\ &= \Omega_X xy + \Omega_Y(1+x^2) - \Omega_Z y + (T_X - xT_Z)(k_x x + k_y y + k_z) \\ &= \Omega_Y + T_X k_z + x(k_x T_X - k_z T_Z) + y(-\Omega_Z + k_y T_X) + x^2(\Omega_Y - k_x T_Z) + xy(-\Omega_X - k_y T_Z) \\ &= a_1 + a_2 x + a_3 y + a_y x^2 + a_8 xy \end{aligned} \tag{507}$$

Similarly for β we have

$$\beta = a_4 + a_5 x + a_6 y + a_7 xy + a_8 y^2 \tag{508}$$

where

$$a_1 = \Omega_Y + k_z T_X \tag{509}$$
$$a_2 = k_x T_X - k_z T_Z \tag{510}$$
$$a_3 = -\Omega_Z + k_y T_X \tag{511}$$
$$a_4 = \Omega_Z + k_z T_Y \tag{512}$$
$$a_5 = \Omega_Z + k_x T_Y \tag{513}$$
$$a_6 = k_y T_Y - k_z T_Z \tag{514}$$
$$a_7 = \Omega_Y - k_x T_Z \tag{515}$$
$$a_8 = -\Omega_X - k_y T_Z \tag{516}$$

These parameters are fundamental and are known as the Ψ transformation. If there is only one object that is moving, it is able to determine optical flow parameters α and β from object parameters. For the case of multiple objects, we just cluster u and v for each object and track across the frame; however, in order to this successfully, we need to individually track different objects. In other words, different objects will create different flow fields. Adiv accounts for this problem by means of segmentation of each surface followed by calculation for all parameters.

One alternative method for structure from motion and optical fields is by estimating the focus of estimation by means of Fourier techniques. Recall that

$$u(x,y) = (-t_x + xt_z)g(x,y) + xy\omega_x - (1+x^2)\omega_y + y\omega_z \tag{517}$$
$$v(x,y) = (-t_y + yt_z)g(x,y) - xy\omega_y + x\omega_z \tag{518}$$

Define the focus of expansion as $(x_f, y_f) \triangleq (t_x/t_z, t_y/t_z)$. In the case when motion is forward, then the motion is always on the image plane. Let $h(x,y) = t_z/Z(x,y)$, known as the scaled inverse depth map. We can then write our equations above as

$$u(x,y) = (x - x_f)h(x,y) + xy\omega_x - (1+x^2)\omega_y + y\omega_z \tag{519}$$
$$v(x,y) = (y - y_f)h(x,y) + (1+y^2)\omega_x - xy\omega_y - x\omega_z \tag{520}$$

We can optimize the problem with a least squares approach, which results in the simplification that a total of 43 FFTs can yield all necessary information to determine structure (Extracting Structure from Optical Flow Using the Fast Error Search Technique, Srinivasan).

Estimating Motion and Structure from Optical Flow

If noise is assumed, then it is possible to estimate the variance by determining the Cramer-Rao bound. Recall that α and β are functions of Z and motion parameters. By include noise, we have

$$\alpha = f_1(\mathbf{T}, \mathbf{\Omega}, Z) + n \tag{521}$$

$$\beta = f_2(\mathbf{T}, \mathbf{\Omega}, Z) + n \tag{522}$$

where n is noise. Assume that $n \sim \mathcal{N}(0, \sigma^2)$. Then we can write the probability distribution $P_1(\alpha|\mathbf{T}, \mathbf{\Omega}, Z)$ and $P_2(\beta|\mathbf{T}, \mathbf{\Omega}, Z)$, where $\mathbf{T}, \mathbf{\Omega}, Z$ are motion and structure parameters. From here, we can derive the Cramer-Rao bounds to analyze the stability and sensitivity of these distributions. It is then possible to expand the optical flow formulation to multiple frames, using the approach derived by Srinivisan (43 FFTs) and track the covariances for a pair of frames using a Kalman filter.

A Discrete Approach to Optical Flow

Assume that for 2 frames we have two points P and P':

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \quad P' = \begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} \tag{523}$$

Between the two frames, assuming only rotation and translation,

$$P' = RP + T \tag{524}$$

R is the rotation matrix and T is the translation vector. There are at least 9 different representations for rotations (e.g. Euler angles, direction cosines, quaternions). The rotation matrix R can be written as

$$R = \begin{pmatrix} 1 & -\Omega_Z & \Omega_Y \\ \Omega_Z & 1 & -\Omega_X \\ -\Omega_Y & \Omega_X & 1 \end{pmatrix} \tag{525}$$

Let (x, y) be a point in the image plane in frame 1 and (x', y') be a point in the image plane in frame 2. Recall that from the perspective equations $x' = X'/Z'$. Substituting our matrices into $P' = RP + T$ yields

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = \begin{pmatrix} 1 & -\Omega_Z & \Omega_Y \\ \Omega_Z & 1 & -\Omega_X \\ -\Omega_Y & \Omega_X & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} T_X \\ T_Y \\ T_Z \end{pmatrix} \tag{526}$$

Next, we substitute into our relations from the perspective/projection equations. In this case,

$$x' = \frac{X'}{Z'} = \frac{X - \Omega_Z Y + \Omega_Y Z + T_X}{-\Omega_Y X + \Omega_X Y + Z + T_Z} \tag{527}$$

Dividing through by Z

$$x' = \frac{\frac{X}{Z} - \frac{\Omega_Z Y}{Z} + \Omega_Y + \frac{T_X}{Z}}{-\Omega_Y \frac{X}{Z} + \Omega_X \frac{Y}{Z} + 1 + \frac{T_Z}{Z}} = \frac{x - \Omega_Z y + \Omega_Y + T_X/Z}{-\Omega_Y x + \Omega_X y + 1 + T_Z/Z} \tag{528}$$

Likewise for y',

$$y' = \frac{Y'}{Z'} = \frac{\Omega_Z x + y - \Omega_X + T_Y/Z}{-\Omega_Y x + \Omega_X y + 1 + T_Z/Z} \tag{529}$$

For unit time, then $\alpha = x' - x$ and $\beta = y' - y$. Then,

$$\alpha = \frac{-\Omega_X xy + \Omega_Y(1+x^2) - \Omega_Z y + (T_X - T_Z x)/Z}{1 + \Omega_X y - \Omega_Y x + T_Z/Z} \tag{530}$$

$$\tag{531}$$

If we assume that $||T_Z/Z|| << 1$, Ω_X and Ω_Y are small. Then

$$\alpha \approx -\Omega_X xy + \Omega_Y(1+x^2) - \Omega_Z y + (T_X - T_Z x)/Z \tag{532}$$

Notice that this is exactly what we had for optical flow. We are strictly talking about image plane displacements. Optical flow is dense, though here, we need to determine α and β for every point. Here, we made no assumptions. In the case of a planar patch, we would yield the same simplification as above for optical flow. Next, we consider, for the discrete case, the rigid planar path $ax + by + cz = 1$ (Tsai and Huang).

Passive Navigation - Translation

Now we consider the case of pure translation. Consider two frames, as in the case, the dilation between the two frames cannot be solved from a single optical flow equation. Therefore, in order to determine camera motion, we require some other optimization. In this case, we use a linear least squares approach to determine a single surface Z that is optimized from the movement parameters we determine. Our optimization problem is:

$$\arg\min_Z \iint \left(\alpha - \frac{T_X - xT_Z}{Z}\right)^2 + \left(\beta - \frac{T_Y - yT_Z}{Z}\right)^2 dxdy \tag{533}$$

We next differentiate the integrand with respect to Z that minimizes the integrand at every point $(x,y)_i$. Let $\alpha_1 = (T_X - xT_Z)$ and $\beta_1 = (T_Y - yT_Z)$. Then

$$\begin{aligned} &\arg\min_Z \iint \left(\alpha - \frac{\alpha_1}{Z}\right)^2 + \left(\beta - \frac{\beta_1}{Z}\right)^2 dxdy \\ &= \arg\min_Z \left(\alpha - \frac{\alpha_1}{Z}\right)^2 + \left(\beta - \frac{\beta_1}{Z}\right)^2 = \arg\min_Z \iint I dxdy \\ &\implies \frac{\partial I}{\partial Z} = 2\Big(\alpha - \frac{\alpha_1}{Z}\Big)\frac{\alpha_1}{Z^2} + 2\Big(\beta - \frac{\beta}{Z}\Big)\frac{\beta}{Z^2} = 0 \implies Z = \frac{\alpha_1^2 + \beta_1^2}{\alpha\alpha_1 + \beta\beta_1} \end{aligned} \tag{534}$$

Substituting back in Z into our original difference expression yields the following

$$\alpha - \frac{\alpha_1}{Z_1} = \alpha - \alpha_1 \frac{(\alpha\alpha_1 + \beta\beta_1)}{\alpha_1^2 + \beta_1^2} = \frac{\beta_1(\alpha\beta_1 - \beta\alpha_1)}{\alpha_1^+ \beta_1^2} \tag{535}$$

Similarly for β

$$\beta - \frac{\beta_1}{Z_1} = \frac{-\alpha_1(\alpha\beta_1 - \beta\alpha_1)}{\alpha_1^2 + \beta_1^2} \tag{536}$$

Thus we have, for our minimization:

$$\arg\min_{\mathbf{T}} \iint \frac{(\alpha\beta_1 - \beta\alpha_1)^2}{\alpha_1^2 + \beta_1^2} dxdy \tag{537}$$

Now we differentiate with respect to T_x, T_y, and T_z. Let us consider the case of similar surfaces and similar motions. It can be shown that if two flow generate the same optical flow, and we know that the motions are purely translational, then the two surfaces are similar and the two camera motions are similar. Let Z_1 and Z_2 be two surfaces. For translations $\mathbf{T}_1 = (T_X^{(1)}, T_Y^{(1)}, T_Z^{(1)})$ and $\mathbf{T}_2 = (T_X^{(2)}, T_Y^{(2)}, T_Z^{(2)})$, then we know that

$$\frac{u}{v} = \frac{T_X^{(1)} + xT_Z^{(1)}/Z_1}{T_Y^{(1)} + yT_Z^{(1)}/Z_1} = \frac{T_X^{(2)} + xT_Z^{(2)}/Z_1}{T_Y^{(2)} + yT_Z^{(2)}/Z_2} \tag{538}$$

$$\implies (T_X^{(1)} + xT_Z^{(1)})(T_Y^{(2)} + yT_Z^{(2)}) = (T_X^{(2)} + xT_Z^{(2)})(T_Y^{(1)} + yT_Z^{(1)}) \tag{539}$$

$$\implies T_X^{(1)}T_Y^{(2)} + xT_Y^{(2)}T_Z^{(1)} + yT_X^{(1)}T_Z^{(2)} + xyT_Z^{(1)}T_Z^{(2)} = T_X^{(2)}T_Y^{(1)} + xT_Y^{(1)}T_Z^{(2)} + yT_X^{(2)}T_Z^{(1)} + xyT_Z^{(2)}T_Z^{(1)} \tag{540}$$

Thus we see that $T_X^{(1)}T_Y^{(2)} = T_X^{(2)}T_Y^{(1)}$, $T_Y^{(2)}T_Z^{(1)} = T_Y^{(1)}T_Z^{(2)}$, and $T_X^{(1)}T_Z^{(1)} = T_X^{(2)}T_Z^{(2)}$, from which it is clear that Z_2 is a dilation of Z_1. This scaling factor cannot be recovered for optical flow, regardless of the number of points at which flow is known.

Alternatively, we can solve the minimization problem

$$\arg\min_Z \iint \left(\alpha - \frac{T_X - xT_Z}{Z}\right)^2 + \left(\beta - \frac{T_Y - yT_Z}{Z}\right)^2 (\alpha_1^2 + \beta_1^2) dxdy \tag{541}$$

and weight the norm. This allows for a closed form solution, where after differentiating with Z, we obtain

$$Z = \frac{\alpha_1^2 + \beta_1^2}{\alpha\alpha_1 + \beta\beta_1} \tag{542}$$

which results in the minimization problem

$$\arg\min_{\mathbf{T}} \iint (\alpha\beta_1 - \beta\alpha_1)^2 dxdy \tag{543}$$

Now let us consider the rotational case. Recall that if we consider only rotational motion, we have

$$\alpha_R = \Omega_X xy + \Omega_Y(x^+1) + \Omega_Z y \tag{544}$$

$$\beta_R = \Omega_X(y^2 + 1) + \Omega_Y xy + \Omega_Z x \tag{545}$$

In this case, we choose to minimize

$$\arg\min_Z \iint [(\alpha - \alpha_R)^2 + (\beta - \beta)^2] dxdy \tag{546}$$

We then differentiate this with respect to $\mathbf{\Omega}$ in order to obtain the desired result (see Bruss and Horn, Passive Navigation). In the case of general motion, we have

$$\arg\min_Z \iint \left(\alpha - \left(\frac{\alpha_1}{Z} + \alpha_R\right)\right)^2 \left(\beta - \left(\frac{\beta}{Z} + \beta\right)\right)^2 (\alpha_1^2 + \beta_1^2) dxdy \tag{547}$$

Differentiating with respect to Z and solving

$$Z = \frac{\alpha_1^2 + \beta_1^2}{(\alpha - \alpha_R)\alpha_1 + (\beta - \beta_R)\beta_1} \tag{548}$$

We then minimize this via the method of Lagrange Multipliers:

$$\iint [(\alpha - \alpha_R)\beta_1 - (\beta - \beta_R)\alpha_1]^2 dxdy + \lambda(T_X^2 + T_Y^2 + T_Z^2 - 1) \tag{549}$$

In other words we solving with the constraint of unit translation. The Bruss and Horn only discusses rotation and general motion. This result is similar to what was presented for Adiv's paper, which is discussed above.

General Motion - Adiv

Consider one surface. For this case, given a set of n flow vectors, we wish to determine the optical Ψ transformation corresponding to this set. The error, with corresponding weights W_i is:

$$\begin{aligned} E = \sum W_i[&(\alpha_i + \Omega_X x_i y_i - \Omega_Y(1-x_i^2) + \Omega_Z y_i - (T_X - xT_Z)/Z_i)^2 \\ &+ (\beta_i + \Omega_X(1+y_i^2) - \Omega_Y x_i y_i + \Omega_Z x_i - (T_Y - yT_Z)/Z_i)^2] \\ &= \sum W_i[(\alpha_i - \alpha_{R_i} - \alpha_{T_i})^2 + (\beta_i - \beta_{R_i} - \beta_{T_i})^2] \end{aligned} \tag{550}$$

We can separate the translational components from Z, so we normalize the translational components. Let $(u_x, u_y, u_z) = (T_X, T_Y, T_Z)/r$ where $r = \sqrt{T_X^2 + T_Y^2 + T_Z^2}$. Furthermore, we will only consider relative depths $\tilde{Z}_i = r/Z_i$, $i = 1, .., n$. For absolute depths, we need stereo. In this notation, we have

$$\alpha_u = u_x - u_z x = \frac{\alpha_T}{\tilde{Z}} \tag{551}$$

$$\beta_u = v_x - v_z y = \frac{\beta_T}{\tilde{Z}} \tag{552}$$

Our overall error takes the form

$$E = \sum W_i[(\alpha_i - \alpha_{R_i} - \alpha_{T_i})^2 + (\beta_i - \beta_{R_i} - \beta_{T_i})^2] = \sum W_i[(\alpha_i - \alpha_{R_i} - \alpha_{u_i}\tilde{Z})^2 + (\beta_i - \beta_{R_i} - \beta_{u_i}\tilde{Z})^2] \tag{553}$$

We now need to find $u_x, u_y, u_z, \Omega_X, \Omega_y, \Omega_Z, \tilde{Z}$, for $i = 1, ..., n$ with the constraint of non-negative depth $\tilde{Z}_i \geq 0$ $\forall$ i. Let us now consider only the optimum value of $\tilde{Z}_i$ as a function of motion parameters. Differentiating g with respect to $\tilde{Z}_i$ yields and soling for $\tilde{Z}_i$

$$\frac{\partial E}{\partial \tilde{Z}_i} = 2W_i[-(\alpha_i - \alpha_{R_i})\alpha_{u_i} - (\beta_i - \beta_{R_i})\beta_{u_i} + (\alpha_{u_i}^2 + \beta_{u_i}^2)\tilde{Z}_i] = 0 \tag{554}$$

$$\implies \tilde{Z}_i = \frac{(\alpha_i - \alpha_{R_i})\alpha_{u_i} + (\beta_i - \beta_{R_i})\beta_{u_i}}{\alpha_{u_i}^2 + \beta_{u_i}^2} \tag{555}$$

Notice that this is very similar to the expression derived by Bruss and Horn. Next, we substitute $\tilde{Z}_i$ back into the objective function so that the error is now only a function of motion parameters:

$$\begin{aligned} E(\mathbf{U}, \mathbf{\Omega}) &= \sum_{i=1}^{n} W_i E_i \\ E_i &= \frac{[(\alpha_i - \alpha_{R_i})\beta_{u_i} + (\beta_i - \beta_{R_i})\alpha_{u_i}]^2}{\alpha_{u_i}^2 + \beta_{u_i}^2} \end{aligned} \tag{556}$$

As depth is non-negative, then we must have the following form for our error:

$$E_i = \begin{cases} \dfrac{[(\alpha_i - \alpha_{R_i})\beta_{u_i} + (\beta_i - \beta_{R_i})\alpha_{u_i}]^2}{\alpha_{u_i}^2 + \beta_{u_i}^2}, & [(\alpha_i - \alpha_{R_i})\beta_{u_i} + (\beta_i - \beta_{R_i})\alpha_{u_i}] > 0 \\ (\alpha_i - \alpha_{R_i})^2 + (\beta_i - \beta_{R_i})^2, & \text{otherwise} \end{cases} \tag{557}$$

This methodology is popular because it can handle multiple surfaces. In all this, however, we have not discussed existence and uniquness of the solution. We will now consider the discrete approach by Tsai and Huang to account for this.

Discrete Approaches to Optical Flow II

For discrete approaches, we need to solve the so called correspondence problem, in which points are mapped from one frame to the next along the surface. Recall that in optical flow, we assume small motions for brightness constancy. However, for large motions, we need to consider displacement as a function of structure and motion. Let us first consider a rigid planar patch. In frame 1 and frame 2, we have coordinates $[x, y, z]^T$ and $[x', y', z']^T$. We assume that the object is able to deform, rotate, and translate in space. Therefore, the relationship between the two points is

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = S \begin{bmatrix} x \\ y \\ z \end{bmatrix} + R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{558}$$

where S is the deformation gradient, R is the rotation matrix, and $[\Delta x, \Delta y, \Delta z]^T$ is the translation. We can write the deformation matrix and the rotation matrix R in terms of direction cosines as follows:

$$S = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix}, \quad R = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \tag{559}$$

where $\phi_i = n_i\theta$ with θ bein the angle of rotation and (n_1, n_2, n_3) being the axis of rotation. We further add the constraint $n_1^2 + n_2^2 + n_3^2 = 1$. We can combine the rotation and translation into a single matrix

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{560}$$

Recall that from our perspective/projection equations, we can write each (x', y', z') and (x, y, z) in their corresponding image planes, namely (X', Y') and (X, Y), respectively, with relationship $X' = Fx'/z'$. For a rigid plan patch $ax + by + cz = 1$, we can write

$$z = \frac{F}{aX + bY + cF} \tag{561}$$

Now, we can obtain a relationship between image plane coordinates, spatial coordinates, and motion parameters as follows. We can substitute our motion equations into our perspective projection equations to obtain:

$$X' = F\frac{x'}{z'} = F\frac{b_{11}x + b_{12}y + b_{13}z + \Delta x}{b_{31}x + b_{32}y + b_{33}z + \Delta z} \tag{562}$$

As $x = Xz/F$ and $t = Yz/F$ from the perspective/projection equations, then

$$\begin{aligned} X' = F\frac{x'}{z'} &= \frac{b_{11}(Xz/F) + b_{12}(Yz/F) + b_{13}z + \Delta x}{b_{31}(Xz/F) + b_{32}(Yz/F) + b_{33}z + \Delta z} \\ &= \frac{(b_{11} + a\Delta x)X + (b_{12} + b\Delta y)Y + (b_{13} + c\Delta x)F}{(b_31 + a\Delta z) + (b_{32} + b\Delta z) + (b_{33} + c\Delta z)F} \end{aligned} \tag{563}$$

Dividing through by $(b_{33} + c\Delta z)F$ yields the following

$$X' = \frac{a_1X + a_2Y + a_3}{a_7X + x_8Y + 1} \tag{564}$$

Similarly, we can write Y' as

$$Y' = \frac{a_4X + a_5Y + a_6}{a_7X + x_8Y + 1} \tag{565}$$

For the case of a rigid planar patch (i.e. only rotation and no deformation), we know that $b_{31} = -\phi_2$, $b_{31} = \phi_1$, and $b_{33} = 1$.

$$\begin{aligned} a_1 &= \frac{1+a\Delta x}{1+c\Delta z} & a_2 &= \frac{-\phi_3+b\Delta x}{1+c\Delta z} \\ a_3 &= \frac{F(\phi_2+c\Delta x)}{1+c\Delta z} & a_4 &= \frac{\phi_3+a\Delta y}{1+c\Delta z} \\ a_5 &= \frac{1+b\Delta y}{1+c\Delta z} & a_2 &= \frac{F(-\phi_1+c\Delta y)}{1+c\Delta z} \\ a_7 &= \frac{-\phi_2+a\Delta z}{(1+c\Delta z)F} & a_2 &= \frac{\phi_1+b\Delta z}{(1+c\Delta z)F} \end{aligned} \tag{566}$$

These are known as "pure parameters." We still have the same problem, as we are unable to decouple $c\Delta z$ from the motion equations. From these equations, we now need to determine $a_1, ..., a_8$, listed above. Let

$$\Delta f = \sum_{i=1}^{8} \beta_i x_i f \tag{567}$$

where $a_1 = \beta_1 + 1$, $a_5 = \beta_5 + 1$, and $\alpha_i = \beta_i$ for all other i, the difference between two frames is $f(x', y') - f(x, y)$, and all X_i's are:

$$\begin{aligned} X_1 &= X\frac{\partial}{\partial X}, \quad X_2 = Y\frac{\partial}{\partial X} \\ X_3 &= \frac{\partial}{\partial X}, \quad X_4 = X\frac{\partial}{\partial Y} \\ X_5 &= Y\frac{\partial}{\partial Y}, \quad X_6 = \frac{\partial}{\partial Y} \\ X_7 &= -X^2\frac{\partial}{\partial X} - XY\frac{\partial}{\partial Y} \\ X_8 &= -XY\frac{\partial}{\partial X} - Y^2\frac{\partial}{\partial Y} \end{aligned} \tag{568}$$

Note that this is a result from Lie group theory, which is not discussed in detail here. Each of the derivative here can be approximated with a finite difference. Then you can solve for β_i, which will yield α_i. However, this does not guarantee uniqueness of the motion parameters. Let the matrix A contain all components a_i:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & 1 \end{bmatrix} \tag{569}$$

For the general motion case (i.e. unrestricted rotation), the rotation matrix now takes the form:

$$R = \begin{bmatrix} n_1^2 + (1-n_1^2)\cos\theta & n_1 n_2(1-\cos\theta) - n_3\sin\theta & n_1 n_2(1-\cos\theta) + n_2\sin\theta \\ n_1 n_2(1-\cos\theta) + n_3\sin\theta & n_2^2 + (1-n_2^2)\cos\theta & n_2 n_3(1-\cos\theta) - n_1\sin\theta \\ n_1 n_3(1-\cos\theta) - n_2\sin\theta & n_2 n_3(1-\cos\theta) + n_1\sin\theta & n_3^2 + (1-n_3^2)\cos\theta \end{bmatrix} \tag{570}$$

However, terms disappear for small θ. This results in the following, for small θ:

$$R = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \tag{571}$$

Once A is determined, we can take the SVD of A,

$$A = U \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_3 \end{bmatrix}, V^T = U\Gamma V^T \tag{572}$$

If the multiplicity of the singular values of A is two (e.g. $\lambda_1 = \lambda_2 \neq \lambda_3$, then the solution for the motion and geometrical parameters is unique aside from the a common scale factor and

$$R = \lambda_1^{-1} A - \Big(\frac{\lambda_1}{\lambda_3} - \det(U)\det(V)\Big) U_3 V_3^T \qquad \begin{bmatrix}\Delta x \\ \Delta y \\ \Delta z\end{bmatrix} = w^{-1}\Big(\frac{\lambda_3}{\lambda_1} - \det(U)\det(V)\Big) U_3 \tag{573}$$

where w is the scale factor. We have the rotation matrix in terms of eigenvalues and eigenvectors of the SVD of the matrix of pure parameters. Furthermore, it can be shown that if the singular values of A are all distinct $\lambda_1 > \lambda_2 > \lambda_3$, then there are exactly two solutions for the motion and geometrical parameters and

$$R = U \begin{bmatrix} \alpha & 0 & \beta \\ 0 & 1 & 0 \\ -s\beta & 0 & s\alpha \end{bmatrix}, \quad \begin{bmatrix}\Delta x \\ \Delta y \\ \Delta z\end{bmatrix} = w^{-1}\Big[-\beta U_1 + \Big(\frac{\lambda_3}{\lambda_2} - s\alpha\Big) U_3 \Big] \tag{574}$$

Finally, if all the eigenvalues are identical, there there is rotation and no translation, with $R = \lambda_1^{-1} A$. Next, let us consider the structure and motion for a rigid objects with curved surfaces. Recall from the perspective equations, with focal length $F = 1$,

$$X = \frac{x}{z} \qquad X' = \frac{x'}{z'} \tag{575}$$

$$Y = \frac{y}{z} \qquad Y' = \frac{y'}{z'} \tag{576}$$

for a point (x, y, z) in frame, (x', y', z') in frame 2, and corresponding image plane coordinates (X, Y) and X', Y'. Assuming the same setup as previously,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix}\Delta x \\ \Delta y \\ \Delta z\end{bmatrix} \tag{577}$$

with

$$R = \begin{bmatrix} n_1^2 + (1 - n_1^2)\cos\theta & n_1 n_2 (1 - \cos\theta) - n_3 \sin\theta & n_1 n_2 (1 - \cos\theta) + n_2 \sin\theta \\ n_1 n_2 (1 - \cos\theta) + n_3 \sin\theta & n_2^2 + (1 - n_2^2)\cos\theta & n_2 n_3 (1 - \cos\theta) - n_1 \sin\theta \\ n_1 n_3 (1 - \cos\theta) - n_2 \sin\theta & n_2 n_3 (1 - \cos\theta) + n_1 \sin\theta & n_3^2 + (1 - n_3^2)\cos\theta \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \tag{578}$$

Now, relating the two equations as before, we have

$$X' = \frac{x'}{z'} = \frac{r_1 x + r_2 y + r_3 z + \Delta x}{r_7 x + r_8 y + r_9 z + \Delta z} = \frac{(r_1 X + r_2 Y + r_3) z + \Delta x}{(r_7 X + r_8 Y + r_9) z + \Delta z} \tag{579}$$

Likewise

$$Y' = \frac{(r_4 X + r_5 Y + r_6) z + \Delta x}{(r_7 X + r_8 Y + r_9) z + \Delta z} \tag{580}$$

We can get z from these two equations and equate them. From our equation for X' and Y', respectively, we have

$$z = \frac{\Delta x - \Delta z X'}{X'(r_7 X + r_8 Y + r_9) - (r_1 X + r_2 Y + r_3)} \tag{581}$$

$$z = \frac{\Delta y - \Delta z Y'}{Y'(r_7 X + r_8 Y + r_9) - (r_4 X + r_5 Y + r_6)} \tag{582}$$

This implies

$$\begin{aligned} &\frac{\Delta x - \Delta z X'}{X'(r_7 X + r_8 Y + r_9) - (r_1 X + r_2 Y + r_3)} = \frac{\Delta y - \Delta z Y'}{Y'(r_7 X + r_8 Y + r_9) - (r_4 X + r_5 Y + r_6)} \\ &\implies (\Delta x - \Delta z X')(Y'(r_7 X + r_8 Y + r_9) - (r_4 X + r_5 Y + r_6)) \\ &\quad = (\Delta y - \Delta z Y')(X'(r_7 X + r_8 Y + r_9) - (r_1 X + r_2 Y + r_3)) \end{aligned} \tag{583}$$

Notice that with further algebraic manipulation, we have:

$$\begin{aligned}&\Delta x(Y'(r_7X + r_8Y + r_9) - (r_4X + r_5Y + r_6)) + \Delta xX'(r_4X + r_5Y + r_6)\\&= \Delta y(X'(r_7X + r_8Y + r_9) - (r_1X + r_2Y + r_3)) + \Delta zY'(r_1X + r_2Y + r_3)\end{aligned} \tag{584}$$

From this equation, we see that there are only terms in $X'X$, $X'Y$, XY', abd YY', with no terms in $X'Y'$. Thus, we can condense this into the following matrix equation:

$$\begin{bmatrix}X' & Y' & 1\end{bmatrix} E \begin{bmatrix}X\\Y\\1\end{bmatrix} = 0 \tag{585}$$

Relating these two equations yields teh following form of E:

$$\boxed{E = \begin{bmatrix}\Delta zr_4 - \Delta yr_7 & \Delta zr_5 - \Delta yr_8 & \Delta zr_6 - \Delta yr_9\\ \Delta xr_7 - \Delta zr_1 & \Delta xr_8 - \Delta zr_2 & \Delta xr_9 - \Delta zr_3\\ \Delta yr_1 - \Delta xr_4 & \Delta yr_2 - \Delta xr_5 & \Delta yr_3 - \Delta xr_6\end{bmatrix} = \begin{bmatrix}e_1 & e_2 & e_3\\ e_4 & e_5 & e_6\\ e_7 & e_8 & e_9\end{bmatrix}} \tag{586}$$

Here E is known as the essential matrix, which contains the essential parameters as a linear function of $\Delta x, \Delta y$, and Δz. This easy to solve as we only have 8 parameters. We normalize the matrix E such that $e_9 = 1$. Next, we do an SVD on E with $E = U\Lambda B^T$, where Λ is a diagonal matrix singular vales. $UU^T = I$ and $VV^T = I$. In this case, there are two solutions for R, namely

$$R = U \begin{bmatrix}0 & -1 & \\ 1 & 0 & \\ & & s\end{bmatrix} V^T = U \begin{bmatrix}0 & 1 & \\ -1 & 0 & \\ & & s\end{bmatrix} V^T \tag{587}$$

where $s = \det(U)\det(V)$. There is only one solution for the translation vector:

$$\begin{bmatrix}\Delta x\\ \Delta y\\ \Delta z\end{bmatrix} = \alpha \begin{bmatrix}\phi_1^T\phi_2/\phi_2^T\phi_3\\ \phi_1^T\phi_2/\phi_1^T\phi_3\\ 1\end{bmatrix} \tag{588}$$

where ϕ_i is the ith row of the matrix E for $i = 1, 2, 3$, and α is some scale factor. The point here is that, for a general structure from motion problem, we have estimated the rotational and translational components. Note that Z is determined from our 2 equations above. This is a linear solution and is known as the 8 point algorithm. However, we have not determined how to obtain the matrix E. Given 8 image point correspondences, it follows from

$$\begin{bmatrix}X' & Y' & 1\end{bmatrix} E \begin{bmatrix}X\\Y\\1\end{bmatrix} = 0 \tag{589}$$

that

$$\begin{bmatrix}X_1X_1' & X_1'Y_1 & X_1' & Y_1'X_1 & Y_1'Y_1 & Y_1' & X_1 & Y_1\\ X_2X_2' & X_2'Y_2 & X_2' & Y_2'X_2 & Y_2'Y_2 & Y_2' & X_2 & Y_2\\ X_3X_3' & X_3'Y_3 & X_3' & Y_3'X_3 & Y_3'Y_3 & Y_3' & X_3 & Y_3\\ X_4X_4' & X_4'Y_4 & X_4' & Y_4'X_4 & Y_4'Y_4 & Y_4' & X_4 & Y_4\\ X_5X_5' & X_5'Y_5 & X_5' & Y_5'X_5 & Y_5'Y_5 & Y_5' & X_5 & Y_5\\ X_6X_6' & X_6'Y_6 & X_6' & Y_6'X_6 & Y_6'Y_6 & Y_6' & X_6 & Y_6\\ X_7X_7' & X_7'Y_7 & X_7' & Y_7'X_7 & Y_7'Y_7 & Y_7' & X_7 & Y_7\\ X_8X_8' & X_8'Y_8 & X_8' & Y_8'X_8 & Y_8'Y_8 & Y_8' & X_8 & Y_8\end{bmatrix} \begin{bmatrix}e_1\\e_2\\e_3\\e_4\\e_5\\e_6\\e_7\\e_8\end{bmatrix} = \begin{bmatrix}-1\\-1\\-1\\-1\\-1\\-1\\-1\\-1\end{bmatrix} \tag{590}$$

However, here we are not considering where the points must lie in the plane in order to ensure a necessary solution. It can be shown that if

$$\begin{bmatrix} X' & Y' & 1 \end{bmatrix} E \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = 0 \tag{591}$$

is satisfied by the image point correspondences of a group of object points not lying on two planes with one plane containing the origin, nor on a cone containing the origin, then the matrix above must be skew symmetric. When noise is added to the image, what contributes to error is the localization of points. Furthermore, the more points, the less the error. Therefore, a simple two frame based problem is sensitive to error in feature (point) extraction and matching points across frames (Liu et. al - real experiments with a robot arm). One method to account for the error is to increase the number of frames considered, known as the long sequence based method.

Recursive 3-D Motion Estimation from a Monocular Image Sequence

Let some point in 3D space with coordinates (x, y, z) have 2D coordinates $P = (X, Y)$. The perspective equation with noise for this case is and assuming focal length $F = 1$ is:

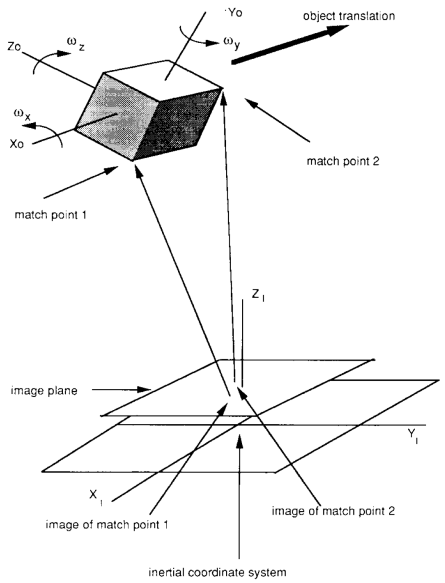
$$X = \frac{x}{z} + n_x \tag{592}$$

$$Y = \frac{y}{z} + n_y \tag{593}$$

where n_x and n_y are the noise components. Then we can write, in vector-matrix form

$$P = \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} x/z \\ y/z \end{pmatrix} + \begin{pmatrix} n_x \\ n_y \end{pmatrix} h[s] + n \tag{594}$$

where $s = (x, y, z)$. Here, the measurement equation for our Kalman filter is simply $h[s] + n$. In general, a Kalman filter as a state model and a measurement model, with the state model corresponding with how the object moves in space. Effectively, we are modeling an object in a camera-centric system with coordinates that vary as a function of time, as shown below:



Now, we will discretize this and determine the appropriate equations to characterize the dynamics. First, we introduce the following notation based on this figure. Let $s_{io} = (x_i, y_i, z_i)^T$ be the object centered coordinates of match point i. Now let $s_R(t) = (x_R(t), y_R(t), z_R(t))^T$ be the camera cantered coordinates of the origin of the translating object reference frame, which we do not observe. Finally, let $s_i(t)$ be the spatial camera centered

coordinate of match point i. Assuming the object undergoes rotation by some matrix R and translation ($s_R(t)$), then

$$s_i(t) = s_R(t) + R(t)s_{io} \tag{595}$$

Now at some time t_k, we have

$$p_i(t_k) = h[s_i(t_k)] + n(t - K) \tag{596}$$

where, as above, $h[t_k]$ is the central projection. Now

$$\begin{pmatrix} X_i \\ Y_i \end{pmatrix}_k = h[s_R(t) + R(t)s_{io}] + n(t_k) \tag{597}$$

for $i = 1, ..., M$ match points. We are now relating image plane coordinates as a function of translation, rotation, and structure. We next require models of translation and rotation. In this case, assuming that motion can be modeled only up to the nth derivative, typically only the second derivative ($n = 2$) in most applications,

$$s_R(t) = s_R(t_0) + \sum_{k=1}^{n} \left. \frac{\partial^{(k)} s_R(t)}{\partial t^{(k)}} \right|_{t=t_0} \frac{(t - t_0)^k}{k!} \tag{598}$$

Now, because we have rotation that is dependent on motion, so we introduce the quaternions $\mathbf{q}(t) = (q_1(t), q_2(t), q_3(t), q_4(t))^T$. In this notation, we have the following form for the rotation matrix

$$R = \begin{pmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_2 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{pmatrix} \tag{599}$$

For constant velocity, quaternions allows for a closed form expression for motion propagation. In terms of direction cosines, the quaternions are

$$q_1 = n_1 \sin\theta/2 \tag{600}$$
$$q_2 = n_2 \sin\theta/2 \tag{601}$$
$$q_3 = n_3 \sin\theta/2 \tag{602}$$
$$q_4 = \cos\theta/2 \tag{603}$$

where (n_1, n_2, n_3) represents the axis of rotation and θ the angle of rotation. The unit quaternion with $||q|| = 1$ is also an appropriate representation. Relating quaternions to velocity, we know that

$$\dot{q}(t) = \Omega(\omega_t)q(t) \quad q(t_0) = v_0 \tag{604}$$

which is a first order differential equation. Here

$$\Omega(\omega_t) = \frac{1}{2} \begin{pmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{pmatrix} \tag{605}$$

The solution to this ODE is

$$\mathbf{q}(t) = \exp[\Omega(t - t_0)]\mathbf{q}(t_0) \tag{606}$$

Now we have a state equation and a measurement model, which requires the Kalman filter framework to solve. Here the state vector is:

$$\mathbf{s}(t) = \begin{pmatrix} x_R(t)/z_R(t) \\ y_R(t)/z_R(t) \\ \dot{x}/z_R(t) \\ \dot{y}/z_R(t) \\ \dot{(z)}/z_R(t) \\ q_1(t) \\ q_2(t) \\ q_3(t) \\ q_4(t) \\ \omega_x \\ \omega_y \\ \omega_z \\ x_1/z_R(t) \\ y_1/z_R(t) \\ z_1/z_R(t) \\ \vdots \\ x_M/z_R(t) \\ y_M/z_R(t) \\ z_M/z_R(t) \end{pmatrix} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ s_8 \\ s_9 \\ s_{10} \\ s_{11} \\ s_{12} \\ s_{13} \\ s_{14} \\ s_{15} \\ \vdots \\ s_{3M+10} \\ s_{3M+11} \\ s_{3M+12} \end{pmatrix} \tag{607}$$

with origin $s_R(t) = (x_R(t), y_R(t), z_R(t))^T$. There are $3M$ structure parameters with a total of $3M+11$ parameters. Without noise, this can be reduced to M parameters. For a Kalman filter framework, we require $s(t) = f[s(t)]+n$, where n is noise. We can now expand the measurement equation $p(t_k) = h[s(t_k)] + v(t_k)$. Recall that for a continuous time model, the Kalman filter has

$$\dot{\mathbf{x}} = F\mathbf{x}(t) + G_t\mathbf{w}_t \tag{608}$$

In the discrete case

$$x(k+1) = \Phi_{k+1,k}x(k) + G_k w_k \tag{609}$$

where $\Phi_{k+1,k} = \exp[(t_{k+1} - t_k)F$. The measurement model is

$$z(k) = H(k)x(k) + v(k) \tag{610}$$

x is the state vector, discussed above. The Kalman filter has the following update:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}(k|k-1) + K(k)[\mathbf{z}(k) - H(k)\hat{\mathbf{x}}(k|k-1)] \tag{611}$$

where $K(k)$ is the Kalman gain:

$$K(k) = P(k|k-1)H(k)^T \times [H(k)P(k|k-1)H(k)^T + R_k]^{-1} \tag{612}$$

For a linear system, this converges. We can also determined the variance and covariance of the estimate:

$$P(l|k-1) = \Phi_{k,k-1}P(k-1|k-1)\Phi_{k,k-1}^T + G_k Q_k G_k^T \tag{613}$$

Finally, the time update comes from the state equation

$$\hat{x}(k+1|k) = \Phi_{k+1,k}\hat{x}(k|k) \tag{614}$$

For this problem, we need to solve a non-linear extending Kalman filter, which involves linearizing around the estimate with a first derivative Taylor approximation

$$H^x(k) = \left.\frac{\partial h[x]}{\partial x}\right|_{x=\hat{x}(k|k-1)} \tag{615}$$

Effectively, for our Kalman gain and covariance, the Jacobian $H^x(k)$ appears in our equations. For extremely non-linear problems, we need to use what is known as the iterated extended Kalman Filter (IEKF). Pentan next simplified the methodology by decreasing the number of state parameters. Here, we shift the central projection along the inverse focal length $\beta = 1/f$. So, the central projection equation becomes

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} X_c \\ Y_c \end{pmatrix} \frac{f}{X_c} \tag{616}$$

We can drop the structure state vector to order N compared to $3N$, as before, only if there is no noise. In this case, we have the following motion equation, with (X, Y, Z) representing the 3D coordinates and (u, v) representing the image plane coordinates. Recall that in the original formulation above, we have $u = fX/Z$ and $v = fY/Z$. For the case of the origin coordinate system being fixed on the image plane, then we have

$$u = \frac{fX}{f + Z} \tag{617}$$

$$v = \frac{fY}{f + Z} \tag{618}$$

If $\beta = 1/f$, then

$$u = \frac{X}{1 + Z\beta} \tag{619}$$

$$v = \frac{Y}{1 + Z\beta} \tag{620}$$

Rewriting, we have

$$X = u(1 + Z\beta) \tag{621}$$

$$Y = v(+Z\beta) \tag{622}$$

$$Z = Z \tag{623}$$

In vector-matrix notation, this equation becomes

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} u \\ v \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} u\beta \\ v\beta \\ 1 \end{pmatrix} \tag{624}$$

where $\alpha = Z$. If we track N points, then our structure is $\alpha_1, ..., \alpha_n$. This is only valid when we have no noise. Now our state vector is only $\mathbf{x} = (t_x, t_y, t_z, \beta, \omega_x, \omega_y, \omega_z, \beta, \alpha_1, ..., \alpha_N)$, which significantly reduces the number of parameters required for a total of $N + 7$ parameters. Note that in the case of noise, we have

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} (1 + \alpha\beta)(u + b_u) \\ (1 + \alpha\beta)(v + b_v) \\ \alpha \end{pmatrix} \tag{625}$$

where b_u and b_v are noise terms. We can write the estimate

$$\begin{pmatrix} \alpha \\ b_u \\ b_v \end{pmatrix} = \begin{pmatrix} z \\ \dfrac{X}{1 + Z\beta} - u \\ \dfrac{Y}{1 + Z\beta} - v \end{pmatrix} \tag{626}$$

We therefore cannot make the assumptions we made by Pentlan in the case of noise. In Pentlan's formulation, the translational components are given by $(t_x, t_y, t_z\beta)$. For rotation, we use quaternions.

$$R = \begin{pmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_2 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{pmatrix} \tag{627}$$

Interframe rotation parameters $(\omega_x, \omega_y, \omega_z)$ can be used to obtain global rotation parameters q_i. The state vector $\mathbf{x}$ is thus

$$\mathbf{x} = (t_x, t_y, t_z, \beta, \omega_x, \omega_y, \omega_z, \beta, \alpha_1, ..., \alpha_N) \tag{628}$$

At this point, one implements an extended Kalman filter. If there are f frames with N points per frame, then there are a total of $2Nf$ measurements, or constraints. There are a total of $6(f-1)$ motion parameters. As

$$2NF + 1 > 6(f-1) + 3N \tag{629}$$

where $2NF+1$ accounts for the scale constraint. This is good for a batch method but not for a recursive method. For a recursive method, we now only $2N$ constraints, but we have $6+3N$ structure parameters. Pentland modifies the problem by having $2N+1$ constraints and and $6+1+N$ degrees of freedom for motion, the camera (β), and structure, respectively at every frame. As long as $N > 7$, then this will work out.

Introduction to Kalman Filters

State-space models are essentially a notational convenience for estimation and control problems, developed to make tractable what would otherwise be a notationally-intractable analysis. Consider a dynamic process described by an n-th order difference equation (similarly a differential equation) of the form

$$y_{i+1} = a_{0,i}y_i + ... + a_{n-1,i}y_{i-n+1} + u_i, \quad i \geq 0 \tag{630}$$

where $\{u_i\}$ is a zero-mean (statistically) white (spectrally) random "noise" process with autocorrelation

$$E(u_i, u_j) = R_u = Q_i\delta_{ij} \tag{631}$$

and initial values $\{y_0, y_{-1}, ..., y_{-n+1}\}$ re zero-mean random variables with a known $n \times n$ covariance matrix

$$P_0 = E(y_{-j}, y_{-k}), \quad j, k \in \{0, n-1\} \tag{632}$$

We also assume that $E(u_i, y_i) = 0$ for $-n+1 \leq j \leq 0$ and $i \geq 0$. In other words, that the noise is statistically independent from the process to be estimated. Under some other basic conditions (Kailath, Sayed et al. 2000) this difference equation can be re-written as

$$\hat{x}_{i+1} \equiv \begin{bmatrix} y_{i+1} \\ y_i \\ y_{i-1} \\ \vdots \\ y_{i-n+2} \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & \dots & a_{n-2} & a_{n-1} \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} y_i \\ y_{i-1} \\ y_{i-2} \\ \vdots \\ y_{i-n+1} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} u_i \tag{633}$$

where

$$A = \begin{bmatrix} a_0 & a_1 & \dots & a_{n-2} & a_{n-1} \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \tag{634}$$

$$\hat{x}_i = \begin{bmatrix} y_i \\ y_{i-1} \\ y_{i-2} \\ \vdots \\ y_{i-n+1} \end{bmatrix} \tag{635}$$

$$G = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{636}$$

This leads to the state space model

$$x_{i+1} = Ax_i + Gu_i \tag{637}$$
$$y_i = [1, 0, ..., 0]x_i \tag{638}$$

or more generally,

$$x_{i+1} = Ax_i + Gu_i \tag{639}$$
$$y_i = Hx_i \tag{640}$$

The new state is a linear combination of both the previous state and the process noise, whereas the process measurements, or observations, are derived from the internal states. These two equations are known as the process model and the measurement model, respectively. For the observer design, or black box problem, in which cannot directly observe the process, we usually fit a process model

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \tag{641}$$

In addition there is some form of measurement model that describes the relationship between the process state and the measurements. This can usually be represented with a linear expression

$$z_k = Hx_k + v_k \tag{642}$$

The terms w_k and v_k are random variables representing the process and measurement noise, respectively. The Kalman filter is as follows. Given the observation z_k, we need estimate the state vector x_k. One constraint is to minimize the least square error. We are trying to estimate the process $x_k = Ax_{k-1} + Bu_k + w_{k-1}$ with a measurement $z_k = Hx_k + v_k$. Here $W \sim \mathcal{N}(0, Q)$ and $V \sim \mathcal{N}(0, R)$, where Q and R are the process noise covariance and measurement noise covariance, respectively. Let $\hat{x}_k^-$ be the a priori estimate at step k and $\hat{x}_k$ be the a posteriori state estimate given z_l. We can define the errors

$$e_k^- \equiv x_k - \hat{x}_k^- \tag{643}$$
$$e_k \equiv x_k - \hat{x}_k \tag{644}$$

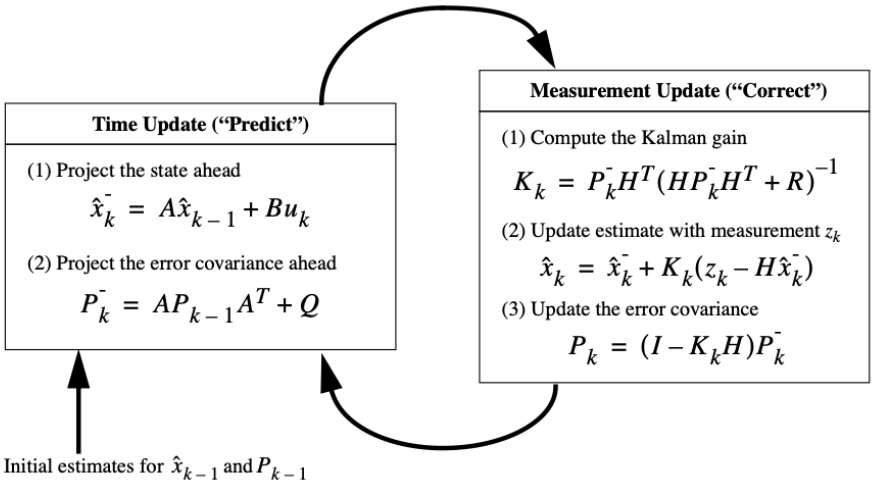
The a priori and a posterior error covariances are , $P_k^- = E[e_k^- e_k^{-^T}]$ and $P_k = E[e_k e_k^T]$, respectively. In deriving the equations for a Kalman filter, we assume that the a posteriori state is a linear combination of the a priori state and measurement, with some proportionality known as the Kalman gain K:

$$\hat{x}_k - \hat{x}_k^- + K(z_k - H\hat{x}_k^-) \tag{645}$$

where $(z_k - H\hat{x}_k^-)$ is known as the residual. The Kalman gain can be determined by minimizing the least square error and has one potential form

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} = \frac{P_k^- H^T}{HP_k^- H^T + R} \tag{646}$$

The meanings of the estimates and Kalman gain come directly from Bayesian probability, with the a posteriori state representing the mean and covariance of the state distribution. A summary for the Discrete Kalman filter implementation is shown below



In the case of structure from motion, we have a non=linear stochastic difference equation, which requires use of an extended Kalman filter. Here, we assume that

$$x_k = f(x_{k-1}, u_k, w_{k-1}) \tag{647}$$

with a measurement $z_k = h(x_k, v_k)$, where w_k and v_k are random variables that represent the process and measurement of noise. Effectively, the process is very similar to Kalman filter and involves linearizing by determining the Jacobian of the non-linear equations. First, we begin by linearizing the estimate

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + Ww_{k-1} \tag{648}$$

$$z_k \approx \tilde{z}_k + H(x_k - \tilde{x}_k) + Vv_k \tag{649}$$

where A, W, H, and V are all Jacobian matrices with forms

$$A_{ij} = \frac{\partial f_i}{\partial x_j}, \quad W_{ij} = \frac{\partial f_i}{\partial w_j}, \quad H_{ij} = \frac{\partial h_i}{\partial x_j}, \quad V_{ij} = \frac{\partial h_i}{\partial v_j} \tag{650}$$

The remainder of the procedure is identical.

The Factorization Method

This method requires an orthographic projection and uses SVD. Let u_{fp}, v_{fp}, where $f = 1, .., F$ and $p = 1, ..., P$, be trajectories of the image plane coordinates. We write the horizontal feature coordinates of u_{fp} into an $F \times P$ matrix U with one row per frame and one column per feature point. Similarly, we can construct a matrix V from the vertical feature coordinate v_{fp} with the same dimensions. We thus have a single $2F \times P$ matrix W

$$W = \begin{bmatrix} U \\ V \end{bmatrix} \tag{651}$$

Next we perform normalization with respect to the average $\bar{a}_{fp}$ and $\bar{b}_{fp}$. Let

$$\tilde{u}_{fp} = u_{fp} - \bar{a}_{fp} \tag{652}$$

$$\tilde{v}_{fp} = v_{fp} - \bar{b}_{fp} \tag{653}$$

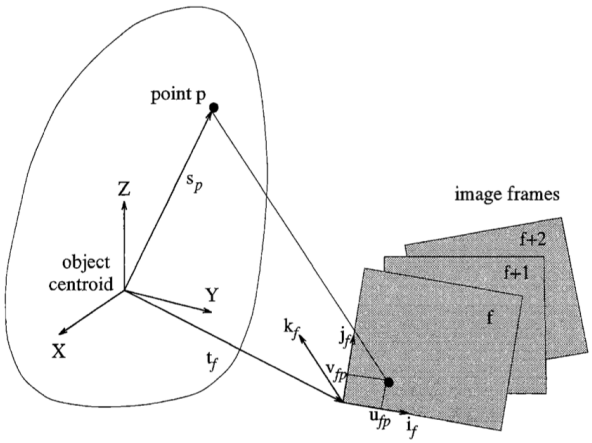
where

$$\bar{a}_{fp} = \frac{1}{P}\sum_{p=1}^{P} u_{fp}, \quad \bar{b}_{fp} = \frac{1}{P}\sum_{p=1}^{P} v_{fp} \tag{654}$$

The rows of V and V are now normalized, or registered, by subtracting from each entry, the mean of the entries within the same row. Now we have

$$\tilde{W} = \begin{bmatrix} \tilde{U} \\ \tilde{V} \end{bmatrix} \tag{655}$$

The following image depicts the two systems of reference used in the problem formulation.



We want to the track the rotation of the frame by looking at the vectors i_f and k_f in the image above at which point we can look at t_f for translation. Effectively our goal is to determine to rank of the matrix $\tilde{W}$. We place the origin of the world reference system at the centroid of the p points $s_p = (x_p, y_p, z_p)^T$, where $p = 1, ..., P$. The orientation is determined by the vectors i_f and k_f (see figure). These are unit vectors that point along the scan lines of the image. We also know that $k_f = i_f \times j_f$, where $\times$ denotes the cross product. Furthermore from the image above we know that u_{fp}, v_{fp} correspond to the image plane coordinates of $s_p = (x_p, y_p, z_p)^T$. With this, we can write the following equation:

$$u_{fp} = i_f^T(r_p - t_f)v_{fp} = f_f^T(r_p - t_f) \tag{656}$$

where $t_f = (a_f, b_f, c_f)^T$ is the vector from the world origin to the origin of the image frame f. Since the origin of the world coordinates is at the centroid of all the 3D points, then

$$\frac{1}{P}\sum_{p=1}^{P} s_p = 0 \tag{657}$$

From our definitions and the formulations above, we can write

$$\tilde{u}_{fp} = u_{fp} - a_f = i_f^T(s_p - t_f) - \frac{1}{P}\sum_{p=1}^{P} i_f^T(s_p - t_f) = i_f^T\left(s_p - \frac{1}{P}\sum_{p=1}^{P} s_p\right) = i_f^T s_p \tag{658}$$

Likewise

$$\tilde{v}_{fp} = j_f^T s_p \tag{659}$$

By concatenating $\tilde{u}_{fp}$ and $\tilde{v}_{fp}$, we now generate $\tilde{W} = RS$, where

$$R = \begin{bmatrix} i_1^T \\ \vdots \\ i_f^T \\ j_1^T \\ \vdots \\ j_f^T \end{bmatrix}, \quad s = \begin{bmatrix} s_1 \\ \vdots \\ s_p \end{bmatrix} \tag{660}$$

Here R is known as the motion matrix, while S is known as the shape matrix. The intuition from creating general matrices for motion comes from Tsai and Huang. In this case, R is a $2F \times 3$ matrix and S is a $3 \times P$ matrix. The rank here, without noise, of the registered matrix $\tilde{W}$ is at most 3. By SVD

$$\tilde{W} = O_1 \Sigma O_2 \tag{661}$$

O_1 is a $2F \times 1$ matrix, Σ is a $P \times P$ diagonal matrix with singular values, and O_2 is a $P \times P$ matrix. Furthermore $O_1^T O_1 = O_2^T O_2 = I$. Σ will have $\sigma_1 > ... > \sigma_p$. We sort the diagonal entries in non-decreasing order. If we only want the the three rows of O_1, the we write $O_1 = [O_1 \quad 'O_1'']^T$ and similarly, we write

$$\Sigma = \begin{bmatrix} \Sigma' & 0 \\ 0 & \Sigma'' \end{bmatrix} \tag{662}$$

where Σ' is of dimension 3×3 and Σ'' is of dimension $P - 3 \times P - 3$. We can also write $O_2 = [O_2 \quad 'O_2'']^T$. When we do this, we can then write

$$O_1 \Sigma O_2 = O_1' \Sigma' O_2' + O_1'' \Sigma'' O_2'' \tag{663}$$

We can only consider the matrix

$$\hat{W} = O_1' \Sigma' O_2' \tag{664}$$

The best possible rank 3 approximation to $\tilde{W}$ is $\hat{W}$. We can write

$$\hat{W} = \hat{R}\hat{S} = (\hat{R}Q)(Q^{-1}\hat{S}) \tag{665}$$

Now, we associate our rotation matrix $R = \hat{R}Q$ and shape matrix $S = Q^{-1}\hat{S}$. To find Q, we use the orthogonal properties as $QQ^{-1} = T$. For both i_f and j_f we can write

$$\hat{i}_F^T Q Q^T \hat{i}_F = I \tag{666}$$

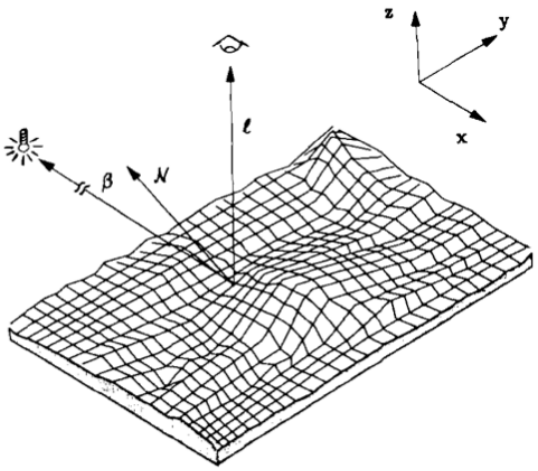
$$\hat{j}_F^T Q Q^T \hat{j}_F = I \tag{667}$$

$$\hat{i}_F^T Q Q^T \hat{f}_F = 0 \tag{668}$$

These are sufficient conditions to determine Q. This algorithm simply involves determining the matrix $\tilde{W} = O_1 \Sigma O_2$ by SVD, followed by defining $\hat{R} = O_1' \Sigma^{1/2}$ and $\hat{S} = \Sigma^{1/2} O_2'$. We then compute Q, followed by the rotation and shape matrices from Q.

Shape from Shading

Shape from shading allows for determination of the depth map by relating the observed intensity with derivative of the depth map, which is known as a reflectance map or the image irradiance equation. We begin by first considering the following surface



Let i be the angle between the normal and the illumination source, j be the angle between the viewer and the source, and $e = i + j$, the angle between the viewer and the normal. Let $z(x, y)$ be hte 3D depth map. Let

$$p = \frac{\partial z(x,y)}{\partial x}, \quad q = \frac{\partial z(x,y)}{\partial y} \tag{669}$$

Then the surface normal is given by $[p, q, -1]^T$. Let $[p_s, q_s, -1]^T$ be the vector that points in the direction of the source. Furthermore, let the vector $[0, 0, -1]^T$ be the vector in the direction of the viewer of the viewers. Then we know that

$$\cos(i) = \frac{1 + pp_s + qq_s}{\sqrt{1 + p^2 + q^2}\sqrt{1 + p_s^2 + q_s^2}} \tag{670}$$

The relationship between (p, q) and intensity at (x, y), $E(x, y)$ is written as follows:

$$E(x,y) = R(p,q) = \frac{1 + pp_s + qq_s}{\sqrt{1 + p^2 + q^2}\sqrt{1 + p_s^2 + q_s^2}} \tag{671}$$

This is known as the image irradiance equation. Now given p and q, we need a set of equations to determine z. Let us first briefly review calculus of variations. If we desire to minimize the funcitional

$$I_1(z) = \iint_Q F(x, y, z, z_x, z_y)dxdy \tag{672}$$

we use the Euler-Lagrange equations

$$F_x - \frac{\partial}{\partial x}F_{z_x} - \frac{\partial}{\partial x}F_{z_y} = 0 \tag{673}$$

In the case we have second partial derivatives in our functional, e.g.

$$I_2(z) = \iint_\Omega F\left(x, y, z, z_x, z_y, z_{xx}, z_{xy}, z_{yy}\right) dxdy \tag{674}$$

then our Euler Lagrange equations take the form

$$F_z - \frac{\partial}{\partial x}F_{z_x} - \frac{\partial}{\partial y}F_{z_y} + \frac{\partial^2}{\partial x^2}F_{z_{xx}} + \frac{\partial^2}{\partial x \partial y}F_{z_{xy}} + \frac{\partial^2}{\partial y^2}F_{z_{yy}} = 0 \tag{675}$$

In the case of shape from shading, we also need the second partials to be constrained for the solution to be smooth. Here our functional must be

$$I_3(p,q) = \iint_\Omega F\left(x, y, p, q, p_x, p_y, q_x, q_y\right) dxdy \tag{676}$$

with corresponding Euler-Lagrange equations

$$F_p - \frac{\partial}{\partial x}F_{p_x} - \frac{\partial}{\partial x}F_{p_y} = 0 \tag{677}$$

$$F_q - \frac{\partial}{\partial x}F_{q_x} - \frac{\partial}{\partial x}F_{q_y} = 0 \tag{678}$$

Clearly, we have a non-linear partial differential equation. One method is to use a discrete optimization (without calculus of variations). We start by considering the quantity

$$E(x,y) - R(p(x,y), q(x,y)) \tag{679}$$

Next, we discretize our problem as shown below

$$\min_{p,q} \sum_{i=1}^{n}\sum_{j=1}^{n}(E_{ij} - R(p_{ij}, q_{ij})) \tag{680}$$

We can differentiate with respect to p_{ij} and q_{ij}, which yields

$$(E_{kl} - R(p_{kl}, q_{kl}))R_p(p_{kl}, q_{kl}) = 0 \tag{681}$$

$$(E_{kl} - R(p_{kl}, q_{kl}))R_q(p_{kl}, q_{kl}) = 0 \tag{682}$$

This however will not work as the solution is $E_{kl} = R(p_{kl}, q_{kl})$, which yields a trivial solution. We now add a smoothness constraint. Consider an infinitesimal segment δc of a curve on the surface. The change in z along the segment is given by

$$\delta z = p\delta x + q\delta y \tag{683}$$

The total change in z along the curve is simply the integral namely

$$\int_C pdx + qdy \tag{684}$$

For a closed curve C,

$$\int_C pdx + qdy = 0 \tag{685}$$

We now discretize this problem. Let ϵ be the spacing between picture cells. Now, consider an elementary square path with lower left hand corner (i,j), where i corresponds to x and j corresponds to y. Then, the integral is approximated by approximated the slope along each of the four segments.

$$e_{ij} = \frac{\varepsilon}{2}\left[p_{i,j} + p_{i+1,j} + q_{i+1,j} + q_{i+1,j+1} - p_{i+1,j+1} - p_{i,j+1} - q_{i,j+1} - q_{i,j}\right] \tag{686}$$

We minimize

$$\min \varepsilon^2 \sum_{i=1}^{n}\sum_{j=1}^{m}\left(E_{ij} - R\left(p_{ij}, q_{ij}\right)\right)^2 + \frac{\lambda}{\varepsilon^2}\sum_{i=1}^{n-1}\sum_{j=1}^{m-1} e_{ij}^2 \tag{687}$$

The solution to this optimization

$$p_{ij}^{k+1} = \bar{p}_{ij}^{k} - \tilde{q}_{ij}^{k} + \frac{\varepsilon^2}{\lambda}\left(E_{ij} - R\left(p_{ij}^{k}, q_{ij}^{k}\right)\right) R_p\left(p_{ij}^{k}, q_{ij}^{k}\right) \tag{688}$$

$$q_{ij}^{k+1} = \bar{q}_{ij}^{k} - \tilde{p}_{ij}^{k} + \frac{\varepsilon^2}{\lambda}\left(E_{ij} - R\left(p_{ij}^{k}, q_{ij}^{k}\right)\right) R_q\left(p_{ij}^{k}, q_{ij}^{k}\right) \tag{689}$$

where

$$\bar{p}_{ij} = \frac{1}{4}\left[\left(p_{i+1,j+1} - 2p_{i+1,j} + p_{i+1,j-1}\right) + 2\left(p_{i,j+1} + p_{i,j-1}\right) + \left(p_{i-1,j+1} - 2p_{i-1,j} + p_{i-1,j-1}\right)\right] \tag{690}$$

$$\bar{q}_{ij} = \frac{1}{4}\left[\left(q_{i+1,j+1} - 2q_{i+1,j} + q_{i+1,j-1}\right) + 2\left(q_{i,j+1} + q_{i,j-1}\right) + \left(q_{i-1,j+1} - 2q_{i-1,j} + q_{i-1,j-1}\right)\right] \tag{691}$$

This method did not incorporate any other boundary conditions gives its discrete nature. One very strong boundary condition to consider, however, is the occluding contour. Suppose we have a surface that is touching a vertical plane. We only see the intersection of the surface along the occluding contour. However, along the occluding contour itself p and q diverge to ∞, so this is not used.

If we however do want to use the boundary conditions from the occluding contour, we can define so called stereo graphic coordinates, namely

$$f = \frac{2p}{\sqrt{1 + p^2 + q^2 + 1}} \tag{692}$$

$$g = \frac{2q}{\sqrt{1 + p^2 + q^2 + 1}} \tag{693}$$

Once can easily construct f and g from p and q and vice versa.

Now let us solve the same optimization problem using calculus of variations. Recall that our optimization problem is

$$\min \iint_{\Omega} (E(x,y) - R(f(x,y), g(x,y)))^2 dxdy \tag{694}$$

The second term we need is a smoothness condition. However before we do this, we would like to reparametrize the problem with

$$p = \frac{4f}{4 - f^2 - g^2} \quad \text{and} \quad q = \frac{4g}{4 - f^2 - g^2} \tag{695}$$

Now we include the following the smoothness criteria, in this case, the regularization term

$$\iint_{\Omega} (f_x^2 + f_y^2 + g_x^2 + g_y^2) dxdy \tag{696}$$

Now our functional is

$$F = (E(x,y) - R(f(x,y), g(x,y)))^2 + \lambda(f_x^2 + f_y^2 + g_x^2 + g_y^2) \tag{697}$$

The Euler-Lagrange equation for this situation becomes

$$(E - R)R_f + \lambda\nabla^2 f = 0 \tag{698}$$

$$(E - R)R_g + \lambda\nabla^2 g = 0 \tag{699}$$

where $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$. We can discretize this operator as

$$\left\{\nabla^2 f\right\}_{ij} \approx \frac{1}{\varepsilon^2}\left[\left(f_{i,j+1} + f_{i+1,j} + f_{i,j-1} + f_{i-1,j}\right) - 4f_{i,j}\right] = \frac{4}{\varepsilon^2}(\bar{f}_{ij} - f_{ij}) \tag{700}$$

where $\bar{f}_{ij} = \frac{1}{4}\left[f_{i,j+1} + f_{i+1,j} + f_{i,j-1} + f_{i-1,j}\right]$. From these, we substitute back into our original Euler-Lagrange formulas, which yields

$$f_{ij} = \bar{f}_{ij} + \frac{\varepsilon^2}{4\lambda}\left(E_{ij} - R\left(f_{ij}, g_{ij}\right)\right) R_f\left(f_{ij}, g_{ij}\right) \tag{701}$$

$$g_{ij} = \bar{g}_{ij} + \frac{\varepsilon^2}{4\lambda}\left(E_{ij} - R\left(f_{ij}, g_{ij}\right)\right) R_g\left(f_{ij}, g_{ij}\right) \tag{702}$$

As an iterative equation,

$$f_{ij}^{k+1} = \bar{f}_{ij} + \frac{\varepsilon^2}{4\lambda}\left(E_{ij} - R\left(f_{ij}^k, g_{ij}^k\right)\right) R_f\left(f_{ij}^k, g_{ij}^k\right) \tag{703}$$

$$g_{ij}^{k+1} = \bar{g}_{ij} + \frac{\varepsilon^2}{4\lambda}\left(E_{ij} - R\left(f_{ij}^k, g_{ij}^k\right)\right) R_g\left(f_{ij}^k, g_{ij}^k\right) \tag{704}$$

The next question is determination of depth, z from p, q. Recall that we can write

$$\delta z = p\delta x + q\delta y \tag{705}$$

Integrating along our cruve yields

$$z(x,y) = z(x_0, y_0) + \int_c (pdx + qdy)dxdy \tag{706}$$

We need to formulate this as a calculus of variations problem. We fit a surface z to the components of p and q such that we want to minimize the difference between the gradients of z and p, q.

$$\min \iint_\Omega \left(z_x - p\right)^2 + \left(z_y - q\right)^2 dxdy \tag{707}$$

The Euler Lagrange equation here reduces to

$$\nabla^2 z = p_x + p_y \tag{708}$$

With the same discrete approximation for the Laplacian, we obtain the following iterative equation

$$z_{ij}^{k+1} = \bar{z}_{ij}^k - \frac{\varepsilon}{4}\left(h_{ij} + v_{ij}\right) \tag{709}$$

where

$$\bar{z}_{ij} = \frac{1}{4}\left(z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1}\right) \tag{710}$$

$$h_{ij} = \frac{1}{2}\left(p_{i+1,j} - p_{i-1,j}\right) \tag{711}$$

$$v_{ij} = \frac{1}{2}\left(q_{i+1,j} - q_{i-1,j}\right) \tag{712}$$

For a smooth surface, we require equivalence of the mixed partials, in this case

$$\frac{\partial^2 z}{\partial x \partial y} = \frac{\partial^2 z}{\partial y \partial x} \tag{713}$$

In terms of p and q, we have

$$\frac{\partial p(x,y)}{\partial y} = \frac{\partial q(x,y)}{\partial x} \implies p_y = q_x \tag{714}$$

There are multiple options to incorporate this smoothness condition into the functional. One option is

$$\iint_\Omega \left(E(x,y) - R(p(x,y), q(x,y))\right)^2 + \mu(x,y)\left(p_y - q_x\right) dxdy \tag{715}$$

The Euler-Lagrange equations are

$$(E-R)R_p + \frac{1}{2}\mu_y = 0 \quad \text{and} \quad (E-R)R_q - \frac{1}{2}\mu_x = 0 \tag{716}$$

An alternative funcitonal to optimize is

$$\iint_{\Omega} (E(x,y) - R(p,q))^2 + \lambda \left(p_y - q_x\right)^2 dxdy \tag{717}$$

The Euler equations are now

$$(E-R)R_p + \lambda(p_{yy} - q_{xy}) = 0 \tag{718}$$
$$(E-R)R_q - \lambda(p_{xx} - q_{yx}) = 0 \tag{719}$$

Now, the iterative equations are

$$p_{ij}^{k+1} = \bar{p}_{ij}^{k} - \frac{1}{2}\tilde{q}_{ij}^{k} + \frac{\varepsilon^2}{2\lambda}\left(E_{ij} - R\left(p_{ij}^{k}, q_{ij}^{k}\right)\right) R_q\left(p_{ij}^{k}, q_{ij}^{k}\right) \tag{720}$$

$$q_{ij}^{k+1} = \bar{q}_{ij}^{k} - \frac{1}{2}\tilde{p}_{ij}^{k} + \frac{\varepsilon^2}{2\lambda}\left(E_{ij} - R\left(p_{ij}^{k}, q_{ij}^{k}\right)\right) R_p\left(p_{ij}^{k}, q_{ij}^{k}\right) \tag{721}$$

where

$$\bar{p}_{ij} = \frac{1}{2}\left(p_{i,j+1} + p_{i,j-1}\right) \quad \text{and} \quad \bar{q}_{ij} = \frac{1}{2}\left(q_{i+1,j} + q_{i-1,j}\right) \tag{722}$$

The above formulations all involve different forms the integrability constraint. An alternative approach to enforcing integrability is known as the Frankot-Chellappa method.

A Method for Enforcing Integrability in Shape from Shading Algorithms - The Frankot-Chellappa Algorithm

We represent $z(x,y)$ as a Fourier expansion. Then p and q become a function of the Fourier coefficients. This algorithm assumes that this is not integrable, but using the Fourier series, we find the closest integral form for p and q and repeat. Suppose that we represent the surface $z(x,y)$ by basis functions $\phi(x,y,\omega)$ such that

$$z(x,y) = \sum_{\omega\in\Omega} C(\omega)\phi(x,y,\omega) \tag{723}$$

Differentiating

$$z_x(x,y) = \sum_{\omega\in\Omega} C(\omega)\phi_x(x,y,\omega) \tag{724}$$

$$z_y(x,y) = \sum_{\omega\in\Omega} C(\omega)\phi_y(x,y,\omega) \tag{725}$$

where $\phi_x = \partial\phi/\partial x$ and $\phi_y = \partial\phi/\partial y$. We want to find $C(\omega)$ which minimizes the function

$$d\left\{\left(\hat{z}_x, \hat{z}_y\right), \left(\tilde{z}_x, \tilde{z}_y\right)\right\} = \iint \left[\left(\tilde{z}_x - \hat{z}_x\right)^2 + \left(\tilde{z}_y - \hat{z}_y\right)^2\right] dxdy \tag{726}$$

where $\hat{z}_x$ and $\hat{z}_y$ are from any shape from shading algorithm and may not be integrable. $\tilde{z}_x$ and $\tilde{z}_y$ are integrable. From there, we can get z. In fact, we can reconstruct z from

$$z(x,y) = \sum_{\omega\in\Omega} C(\omega)\phi(x,y,\omega) \tag{727}$$

Suppose we compute $\hat{z}_x(x,y)$ and $\hat{z}_y(x,y)$ such that

$$\hat{z}_x(x,y) = \sum \hat{C}_1(\omega)\phi_x(x,y,\omega) \tag{728}$$

$$\hat{z}_y(x,y) = \sum \hat{C}_2(\omega)\phi_y(x,y,\omega) \tag{729}$$

It is possible to show that the $\tilde{C}(\omega)$ that minimizes

$$d\left\{\left(\hat{z}_x,\hat{z}_y\right),\left(\tilde{z}_x,\tilde{z}_y\right)\right\} = \iint \left[(\tilde{z}_x - \hat{z}_x)^2 + (\tilde{z}_y - \hat{z}_y)^2\right] dxdy \tag{730}$$

is

$$\tilde{C}(\omega) = \frac{P_x(\omega)\hat{C}_1(\omega) + P_y(\omega)\hat{C}_2(\omega)}{P_x(\omega) + P_y(\omega)} \tag{731}$$

where

$$P_x(\omega) = \iint \left|\phi_x(x,y,\omega)\right|^2 dxdy \tag{732}$$

$$P_y(\omega) = \iint \left|\phi_y(x,y,\omega)\right|^2 dxdy \tag{733}$$

Once C is known, we can determine $z(x,y)$ and its derivatives from

$$\tilde{z}(x,y) = \sum_{\omega\in\Omega} \tilde{C}(\omega)\phi(x,y,\omega) \tag{734}$$

$$\tilde{z}_x(x,y) = \sum_{\omega\in\Omega} \tilde{C}(\omega)\phi_x(x,y,\omega) \tag{735}$$

$$\tilde{z}_y(x,y) = \sum_{\omega\in\Omega} \tilde{C}(\omega)\phi_y(x,y,\omega) \tag{736}$$

This is known as projection onto convex space (POCS). We now summarize the shape from shading work up to this point. From the known intensity, we can write a reflectance map

$$I(x,y) = \mathcal{R}(z_x, z_y, \beta, l, \rho) \tag{737}$$

where ρ is the albedo, or the proportion of the incident light or radiation that is reflected by a surface, β is the illumination direction vector, and l is the vector between the surface and camera. Note that albedo is the intrinsic reflectivity of the material. We are trying to determine a surface $z(x,y)$ we are reconstructing using $z_x(x,y)$ and $z_y(x,y)$. So far, we have considered the condition of integrability $z_{xy} = z_{yx}$. Suppose that we constructed a surface where we minimized

$$\iint (I - R(\hat{z}_x,\hat{z}_y))^2 + \lambda(\hat{z}_{xx}^2 + 2\hat{z}_{xy}^2 + \hat{z}_{yy}^2)dxdy \tag{738}$$

Here $\hat{z_x}$ and $\hat{z}_y$ are not necessarily integrable. One option, without imposing an additional constraint is to find the nearest integrable derivatives of $z(x,y)$, namely $\tilde{z}_x = \tilde{z}_y$ with the new integrability condition $\partial\tilde{z}_x/\partial x = \partial\tilde{z}_y/\partial y$. To determine $\tilde{z}$ by minimizing the function

$$d\left\{\left(\hat{z}_x,\hat{z}_y\right),\left(\tilde{z}_x,\tilde{z}_y\right)\right\} = \iint \left[(\tilde{z}_x - \hat{z}_x)^2 + (\tilde{z}_y - \hat{z}_y)^2\right] dxdy \tag{739}$$

We can perform this optimziation by doing a Fourier expansion of $z(x,y)$ and solve for the optical coefficients for the basis functions (see above). In the case of Fourier expansions our integrable surfaces can be written as

$$\tilde{z}(x,y) = \sum_{\omega\in\Omega} \tilde{C}(\omega)\exp(j\omega\cdot(x,y)) \tag{740}$$

The derivatives of the basis functions, as there are Fourier, are $\phi_x = j\omega_x\phi$ and $\phi_y = j\omega_y\phi$. Furthermore note that $P_x \propto \omega_x^2$ and $P_y \propto \omega_y^2$. So we can write

$$\tilde{C}(\omega) = \frac{-j\omega_x \hat{C}_x(\omega) - j\omega_y \hat{C}_y(\omega)}{\omega_x^2 + \omega_y^2} \tag{741}$$

where $\hat{C}_x(\omega)$ and $\hat{C}_y(\omega)$ come from any shape from shading algorithm. Given the Fourier expansion we can also write $\tilde{C}_x(\omega) = j\omega_x \tilde{C}(\omega)$ and $C_y(\omega) = j\omega_y \bar{C}(\omega)$. This is also true for $\tilde{C}_x(\omega)$ and $\tilde{C}_y(\omega)$. At low frequencies, $\tilde{C}(\omega)$, then this solution is not good as there is loss of low frequency information. Suppose we have low resolution data with a high resolution image. We are able to regularize the low frequency data to obtain the solution.

Photometric Stereo

In the case of photometric stereo, we are considering multiple sources of illumination. Our first reflectance mapping equation is

$$I_1(x, y) = R_1(p, q) \tag{742}$$

Suppose we have a second equation

$$I_2(x, y) = R_2(p, q) \tag{743}$$

The reflectance map changes because p_s and p_q, the vectors in the direction source of illumination changes with the new source. We can similarly have $I_3(x, y) = R_3(p, q)$ and so on. Define a vector $\mathbf{I}$ be the vector of illumination values at some arbitrary point (x, y)

$$\mathbf{I} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} \tag{744}$$

Furthermore, we define unit column vectors $\mathbf{n}_1 = (n_{11}, n_{12}, n_{13})^T$, $\mathbf{n}_2 = (n_{21}, n_{22}, n_{23})^T$, and $\mathbf{n}_3 = (n_{31}, n_{32}, n_{33})^T$ defining the directions of incident illumination. We construct the matrix N as

$$\mathbf{N} = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} \tag{745}$$

So we can write

$$\mathbf{I} = \rho \mathbf{N} \mathbf{n} \tag{746}$$

where $\mathbf{n} = (n_1, n_2, n_3)^T$ is the column vector corresponding to the unit surface normal at (x, y). Assume that $\mathbf{N}^{-1}$ exists then

$$\rho \mathbf{n} = \mathbf{N}^{-1} \mathbf{I} \tag{747}$$

which is only true if the vectors $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$ do not lie on a plane. Thus

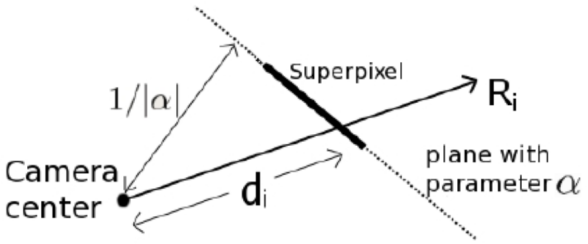
$$\boxed{\mathbf{n} = \frac{1}{\rho} \mathbf{N}^{-1} \mathbf{I}} \tag{748}$$

This generalizes nicely in face analysis in what is known as the illumination cone, in which if we illuminate a face from a single point source, then we need a single set of three basis vectors to fully describe the depth map of the face. Other approaches include motion stereo, in which the motion of an object in space can be used to determine its three-dimensional structure. In general, these are all known as physics based methods.

Make3D: Learning 3D Scene Structure from a Single Still Image

The goal of Make3D is to infer 3D models from monocular cues. In general, we (1) segment into planar patches (super-pixels), (2) build models from depth maps, (3) estimate orientation and location of patches, and (4) construct a 3D model. Markov Random Fields allow for characterizing the context of the pixels within the patches. Properties that can be modeled include a single image (e.g. depth from features, connectedness,

coplanarity), multiple images (e.g. depths from triangulation) and objects (e.g. orientation, relative location, etc.). Essential to Make3D is the superpixel model. The superpixel is a plane. We model it as a 3D mesh of polygons using different segmentation algorithms (e.g. Felzenzwalb and Huttenlocher's segmenter) in order to determine location and orientation of each superpixel.



We want to figure out the plane parameter $\alpha \in \mathbb{R}^3$. $\hat{\alpha} = \alpha/|\alpha|$ is the unit normal of the plane, where $1/|\alpha|$ is the distance from the camera center to the plane. Thus $q^T\alpha = 1$ for any point $q \in \mathbb{R}^3$ on the plane. Consider $R_i \in \mathbb{R}^3$, which is a unit length ray pointing from camera center to pixel i on image plane (using "reasonable guess" of camera's intrinsic parameters). Let $d_i = 1/R_i^T\alpha$ be the distance of point i having ray R_i from the camera center if it lies on the plane described by α. This characterizes the world in terms of planar patches. For an image, this is generalized such that at each point $x_i \in \mathbb{R}^{524}$, filters are run on each superpixel to obtain additional contextual information. Boundary features $\epsilon_{ij} \in \{0, 1\}$ must also be included.

Let $P(y_{ij}|\epsilon_{ij}; \psi)$ model the confidence of superpixels i, j belonging to same planar surface (0 for boundary/fold - 1 for planar). Additionally, $P(\alpha \mid X, v, y, R; \theta)$ models the depth and orientation parameters of superpixels composed of $f_1(\alpha_i \mid X_i, v_i, R_i; \theta)$, which are the plane parameters as a function of the features for a single superpixel i, and $f_2(\alpha_i, \alpha_j \mid y_{ij}, R_i, R_j)$, which are the plane parameters as a function of edge features between superpixels (i, j). f_1 corresponds to the unary term from the Gibbs distribution in the MRF, while f_2 corresponds to the binary term. Furthermore let the distribution $P(v_i \mid x_i; \phi_r)$ model each each pixels ability to predict parameters of the associated superpixel. Just by knowing the binary map, we are able to get an idea of the structure of the object, though this is not sufficient.

The goal of the model is to predict the depth $\hat{d}$ as a function of the features x. We penalize using a relative error $\hat{d}/d - 1$ where $\hat{d} = x^T\theta_r$ and $1/d = T_{i,s_i}^T\alpha_i$ (i.e. d is the real depth). The distribution for f_1 effectively a Gibbs distribution for the features, namely

$$f_1(\alpha_i \mid X_i, v_i, R_i; \theta) = \exp\left(-\sum_{s_i} v_{i,s_i} \left|R_{i,s_i}^T\alpha_i \left(x_{i,s_i}^T\theta_r\right) - 1\right|\right) \tag{749}$$

Parameters are learned from the pseudo log-likelihood of $P(\alpha \mid \cdots)$. Furthermore, since f_2 does not depend on θ_r, we can predict rotation as

$$\theta_r^* = \text{argmin}_{\theta_r} \sum_i \sum_{s_i} v_{i,s_i} \left|\frac{1}{d_i, si}\left(x_{i,s_i}^T\theta_r\right) - 1\right| \tag{750}$$

We also need to determine the confidence of our depth prediction, v. Given a model $\hat{d} = x_i^T\theta_r$ for predicting depth, we can build a model to predict the expected error and learn

$$\frac{\left|d_i - x_i^T\theta_r\right|}{d_i} = \frac{1}{1 + \exp\left(-\phi_r^T x_i\right)} \tag{751}$$

This (ideally) can predict how well a feature predicts the depth of a pixel. Presumably, $v = 1-(1+\exp(-\phi_r^T x_i))^{-1}$ indicates the confidence of prediction ability. We also need to consider superpixel interaction, which is determined by f_2. This is important at the edges of the superpixels. In this case

$$f_2\left(\alpha_i, \alpha_j \mid y_{ij}, R_i, R_j\right) = \prod_{\{s_i, s_j\} \in N} h_{s_i, s_j}\left(\alpha_i, \alpha_j \mid y_{ij}, R_i, R_j\right) \tag{752}$$

s_i, s_i are pixels from superpixels i, j respectively, chosen according to the figure depending on property to be modeled (i.e. connectivity, planarity, linearity). This is where MRFs are very useful. MRFs are very useful, in general, for when there are local patterns, which can be used to describe a global pattern. We use a Gibbs distribution to describe the likelihood for the pattern. Neighboring superpixels tend to be connected if no occlusion. We can use pairs of neighboring pixels (s_i, s_j) along the boundaries of superpixels i and j to determine the function h for f_2,

$$h_{s_i, s_j} = \exp\left(-y_{ij} \frac{\left|d_{i,s_i} - d_{j,s_j}\right|}{\sqrt{d_{i,s_i} d_{j,s_j}}}\right) \tag{753}$$

Additionally, neighboring superpixels tend to belong to the same plane if there is no fold. The penalty for this case is, for a pair of pixels (s_i'', s_j'') in the center of superpixels i and j, respectively is:

$$h_{s_j''} = \exp\left(-y_{ij} \left|\left(R_{j,s_i''}^T \alpha_i - R_{j,s_i''}^T \alpha_j\right) \hat{d}\right|\right) \tag{754}$$

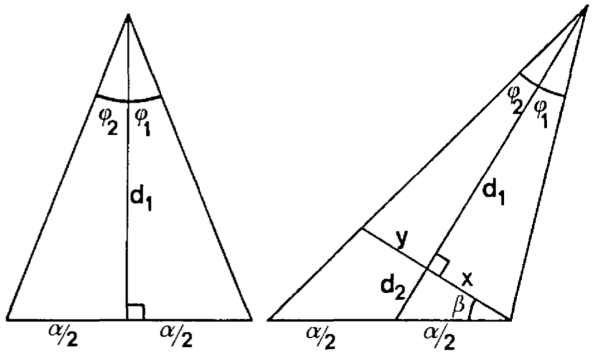
The third criteria, superpixels lying on a straight line are likely to lie on the same plane. We use the same penalty function as above for this case. MRFs allow for the introduction for constraints of motion, geometry, etc. The final step is statistical inference,

$$\begin{aligned} \alpha^* &= \text{argmax}_\alpha \log P\left(\alpha \mid X, v, y, R; \theta_r\right) \\ &= \text{argmax}_\alpha \log \tfrac{1}{Z} \textstyle\prod_i f_1\left(\alpha_i \mid X_i, v_i, R_i; \theta\right) \prod_{i,j} f_2\left(\alpha_i, \alpha_j \mid y_{ij}, R_i, R_j\right) \end{aligned} \tag{755}$$

Each term results in an L1 norm, which is a linear function of α.

Introduction to Stereo

Consider a case when we have a left and right camera, much like our eyes. Fixation on a single point results in some disparity depending on the relative orientation. We quantify the disparity by the angle between the two cameras relative to the object. The image below describes the problem setup.



The left diagram shows the simplest case of an object centered along the axis between the eyes. The interocular distance is denoted by α, and the disparity is given by $\phi = \phi_1 + \phi_2$. The distance of interest is d_1 and is a function of the interocular separation α and the disparity ϕ. The right diagram shows the general case of an object lying off axis. Here, the distance of interest is $d = d_1 + d_2$ is a function of the disparity $\phi_1 + \phi_2$, the interocular separation α and the off axis angle β.

Let us consider the first case, when an object is centered between the two cameras. In the simple case when $\phi_1 = \phi_2$, then

$$d_1 = \frac{\alpha}{2} \cot \phi_1 = \frac{\alpha}{2} \cot \frac{\phi}{2} \tag{756}$$

Now, in the general case

$$d_2 = \frac{\alpha}{2}\sin\beta \quad \text{and} \quad x = \frac{\alpha}{2}\cos\beta \tag{757}$$

From the law of sines

$$x + y = \frac{a\cos(\phi_2 + \beta)}{\cos\phi_2} = a(\cos\beta - \sin\beta\tan\phi_2) \tag{758}$$

Thus

$$y = \frac{a}{2}(\cos\beta - 2\sin\beta\tan\phi_2) \tag{759}$$

As a consequence, we can see that

$$\tan\phi_1 = \frac{\alpha\cos\beta}{2d_1} \quad \text{and} \quad \tan\phi_2 = \frac{a\cos\beta}{2d_1 + 2a\sin\beta} \tag{760}$$

Thus

$$\tan\phi = \frac{\tan\phi_1 + \tan\phi_2}{1 - \tan\phi_1\tan\phi_2} = \frac{a\cos\beta(4d_1 + 2a\sin\beta)}{4d_1^2 + 4d_1\alpha\sin\beta - a^2\cos^2\beta} \tag{761}$$

This yields the following equations for the d_1 and the distance of interest d

$$d_1 = \frac{a}{2}\left(\cos\beta\cot\phi - \sin\beta + \sqrt{1 + \cos^2\beta\cot^2\phi}\right) \tag{762}$$

$$\implies d = d_1 + d_2 = \frac{a}{2}\left(\cos\beta\cot\phi + \csc\phi\sqrt{1 - \sin^2\beta\cos^2\phi}\right) \tag{763}$$

IN the case when $\beta = 0$, corresponding to the simple case

$$d = \frac{\alpha}{2}(\cot\phi + \csc\phi) \tag{764}$$

This is reasonable as if β is too large, then only one camera is able to see it. With the trigonometric identity $\cot\phi + \csc\phi = \cot\phi/2$,

$$d \approx \frac{\alpha}{2}\cot\frac{\phi}{2} \tag{765}$$

We can write then

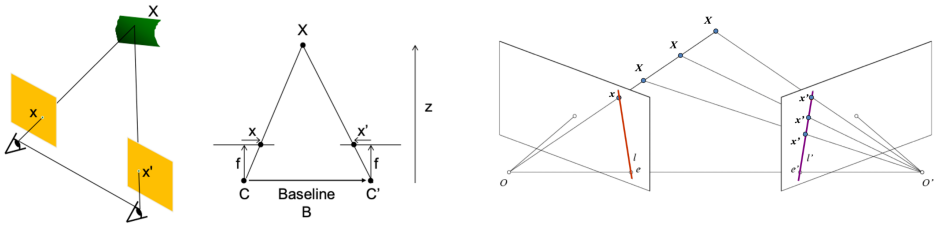
$$d + \Delta d \approx \frac{a}{2}\cot\frac{\phi + \Delta\phi}{2} \implies \frac{\Delta d}{d} \approx \cot\frac{\phi + \Delta\phi}{2}\tan\frac{\phi}{2} - 1 \tag{766}$$

A Taylor series expansion and dropping all higher order terms yields

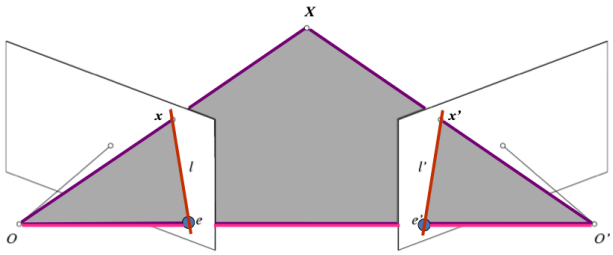
$$\frac{\Delta d}{d} \approx \frac{-\Delta\phi}{\phi + \Delta\phi}\left(1 + \frac{2\phi^2\Delta\phi + \phi(\Delta\phi)^2}{12} + \dots\right) \approx \frac{-\Delta\phi}{\phi + \Delta\phi} \tag{767}$$

The idea of stereo is to find a point on the left, find the corresponding the point on the right and determining the disparity. The problem, however, is matching points, which is the focus of the following.

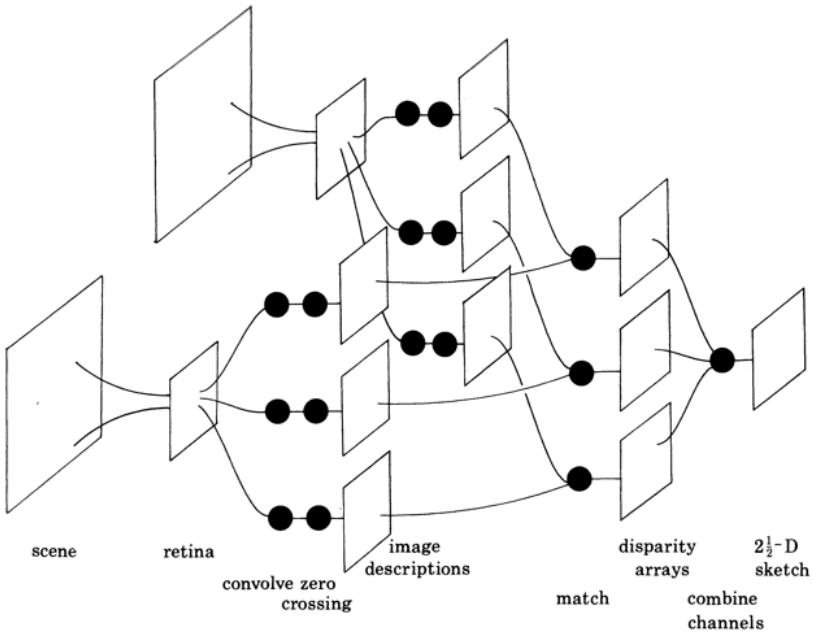
Epipolar geometry relates cameras from two positions, and stereo depth estimation recovers depth from two images. The goal is to recover depth by finding the image coordinate x' that correspond to x, as shown below on the left:



We want to understand this geometry to prevent false correspondences. Consider the case, as in the right where potential matches for x line on line l', while potential matches for x' line on the line l. Let us define the following. The baseline is the line connecting the two camera centerr. The epipoles are the intersections of the baseline with the image planes, or the projections of the other camera center. The epipolar plane is the plane containing the baseline (1D Family). Epipolar lines are intersections of epipolar planes with image planes, which are always in corresponding pairs. The geometry of the system is described in the image below:



We restrict our search to the epipolar lines to prevent false correspondences. We can determine a relationship between x and x' by the so-called Essential matrix E (recall that this is the same matrix derived by Tsai and Huang). The Marr-Pogio model of human stereo is one of the first algorithms for stereo. A schematic diagram for the implementation of a stereo algorithm my Grimson is shown below.



There are multiple approaches to stereo, which are too immense to discuss in detail here. However, the general pipeline involves some Gaussian smoothing followed by image description, matching, determination of disparity arrays and combination of channels. Matching involves the following steps:

(1) Fix the eye position
(2) Locate a zero crossing in one image
(3) Divide the region about the corresponding point in the second image image into three pools
(4) Assign a match to the zero crossing based on the potential matches within the pools
(5) Disambiguate any ambiguous matches
(6) Assign the disparity values to a buffer.

Some criteria for matching zero crossings include: (1) the zero crossings must come from convolutions with the same size filter, (2) the zero-crossings must have the same sign, and (3) the zero crossings must have roughly the same orientation. In general, the objective is to eliminate false matching between points, which is the purpose of such criteria. The denser the features, the more likely to make a mistake. One common method for testing is the random dot stereogram. In general, we can induce a disparity by shifting the same map across itself, which result in artificial depth information depending on the same degree of shifting, as we have difference in the location of matched zero-crossings. For sparse features, we also require some interpolation. The difference for other algorithms is what features are used. Let us first consider Grimson's algorithm more rigorously. Recall that for matching, we require a specific range for to search for zero-crossings between the left and right cameras, namely

$$\{(x', y) \mid x + d_i - w_c \leqslant x' \leqslant x + d_i + w_c\} \tag{768}$$

We also require a continuity constraint. Consider an $n \times n$ image, in which every point can be matched to every other point. In the most basic searching algorithm, we have to consider $(n^2)^{n^2}$ possible matches. The features have a certain density ρ. In this case, we only need to consider $(\rho n^2)^{\rho n^2}$. However, this is still too large a search. The expected density ρ of zero-crossings is $1/cw$, where $c \approx 1.87$. This reduces the possible number of candidates to n^2/cw. Thus the total number of possible correspondences is $(n^2/cw)^{n^2/cw}$. If we restrict our search to only the epipolar lines, then a point on line y can be matched to points on lines v' such that $y - v \leq v' \leq y + v$. Then, we have a space of only $(2v+1)n/cw$ possible matches, with a total number of $(n^2/cw)^{n(2v+1)/cw}$ possible correspondences. So far, we have only considered 1 filter. Consider when we have $k+1$ filters. Then the search is on the order of

$$\left[\frac{n^2}{2^k cw}\right]^{n(2v+1)/2^k cw} \tag{769}$$

The question is whether features persist across k. If this is the case, then now we only have

$$\left[\frac{n^2}{cw_0 2^i}\right]^{(2v+1)} \tag{770}$$

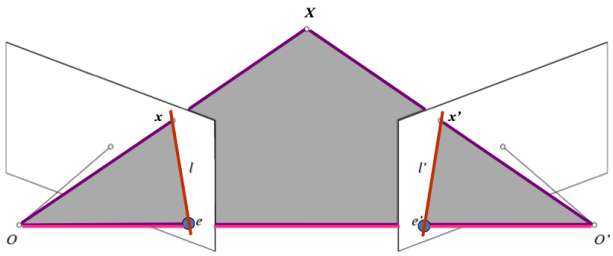
Thus over all scales, the algorithm explores

$$\left[\frac{n^2}{2^k cw}\right]^{n(2v+1)/2kcw} + \sum_{i=0}^{k-1} \left[\frac{n^2}{cw_0 2^i}\right]^{(2v+1)} \tag{771}$$

correspondences, which can be reduced further to

$$\left[\frac{n^2}{cw_0}\right]^{(2v+1)} \left[\frac{1 - \left(\frac{1}{2^{(2v+1)}}\right)^k}{1 - \left(\frac{1}{2^{(2v+1)}}\right)} + \left(\frac{n^2}{cw_0 2^{k(2v+1)}}\right)^{n/cw_0 2k}\right] \tag{772}$$

Let us now mathematically look into stereo algorithms. Recall our epipolar surface,



defined by a translation t and rotation R. We can write

$$x' = R(x - t) \tag{773}$$

The three vectors x, t, x' are coplanar and all line on the epipolar plane. As a result,

$$x^T(t \times x) = 0 \tag{774}$$

We have two constraints, one for rigid motion (i.e. $x' = R(x - t)$)) and coplanarity (i.e. $x^T(t \times x) = 0$). Combining these two,

$$(x - t) = R^{-1}x' \implies (x - t)^T = x'^T(R^{-1})^T \tag{775}$$

As $RR^T = I$,

$$\left(x'^T R\right)(t \times x) = 0 \tag{776}$$

Recall that we can write a cross product between two vectors **a** and **b**,

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{777}$$

Putting it all together, we cn write

$$\begin{aligned} \left(x'^T R\right)(t \times x) &= 0 \\ \left(x'^T R\right)([t_x])x &= 0 \\ x'^T(R[t_x])x &= 0 \\ x'Ex = 0 \quad &\text{where} \quad E = R[t_x] \end{aligned} \tag{778}$$

Note here that $[t_x]$ is matrix decomposition from the vector matrix product. Once we have the essential matrix E, we can determine where the epipolar lines are.

$$l' = Ex \tag{779}$$

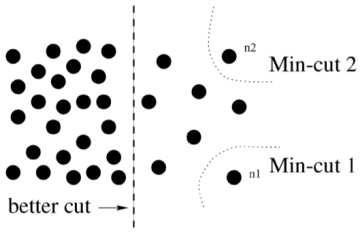
$$l = E^T x' \tag{780}$$

This comes from $x^T l = 0$ and $x'^T l' = 0$. The epipoles are determined from $e'^T E = 0$ or $Ee = 0$. This is known as the 8-point algorithm, as it relates the point in the stereo algorithm through the matrix E, which we match by means of SVD. In the case when don't x and x' exactly, E is multiplied by two linear transformations, resulting in F, known as the fundamental matrix.

Normalized Cuts

Normalized cut is a segmentation algorithm. The normalized cut framework is based on graph theory. Consider a graph $\mathcal{G} = (V, E)$, where V are the vertices and E are the edges. Consider 2 classes at a time. We have to

partition the graph into two disjoint sets A and B such that $A \cup B = V$ and $A \cap B = \emptyset$. We can do this by simply removing edges that connect the two parts. This is called a cut, in graph theory such that



$$\text{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{781}$$

We attempt to find a partition that minimizes the cut, which will yield a segmentation. The problem with this min cut approach is that if some partition is alone, then it is put into its own category. An example of this limitation is shown in the figure to the left. In other words the min cut encourages the creating of small regions. One option to account for this problem is to simply normalize the cut, known as the Normalized cut (Ncut):

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)} \tag{782}$$

where $\text{assoc}(A, V) = \sum_{u \in A, t \in V} w(u, t)$ is the total connection from nodes in A to all nodes in the graph. The small partition problem will disappear as the cut value will almost certainly be a large percentage of the total connection from that small set to other nodes. Let

$$\text{Nassoc}(A, B) = \frac{\text{assoc}(A, A)}{\text{assoc}(A, V)} + \frac{\text{assoc}(B, B)}{\text{assoc}(B, V)} \tag{783}$$

Then it is easy to show that

$$\begin{aligned} \text{Ncut}(A, B) =& \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)} \\ =& \frac{\text{assoc}(A, V) - \text{assoc}(A, A)}{\text{assoc}(A, V)} + \frac{\text{assoc}(B, V) - \text{assoc}(B, B)}{\text{assoc}(B, V)} \\ &= 2 - \left(\frac{\text{assoc}(A, A)}{\text{assoc}(A, V)} + \frac{\text{assoc}(B, B)}{\text{assoc}(B, V)} \right) = 2 - \text{Nassoc}(A, B) \end{aligned} \tag{784}$$

The problem for minimizing the normalized cut is NP complete. The problem was solved with spectral graph theory, which connects graphs with matrices. It can be shown that minimizing the normalized cut is equivalent to the following:

$$\min_x N\,\text{cut}(\boldsymbol{x}) = \min_y \frac{y^T(\mathbf{D} - \mathbf{W})y}{\boldsymbol{y}^T \mathbf{D} y} \tag{785}$$

D is an $N \times N$ diagonal matrix with $d(i)$ on its diagonal where $d(i) = \sum_j w(i, j))$. $y(i) \in (1, -b)$ and $y^T D 1 = 0$. In this case $y = (1 + x) - b(1 - x)$, $b = k/1(1 - k) = \sum_{x_i > 0} d_i / \sum_{x_i < 0} d_i$. $x_i = 1$ if node i in A and -1 otherwise. We have converted the N cut minimization problem to minimizing the ratio of quadratic forms, known as the Rayleigh quotient in classical physics.

Proof: We can rewrite $\text{Ncut}(A, B)$ as

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(B, A)}{\text{assoc}(B, V)} = \frac{\sum_{(x_i > 0, x_j < 0)} -w_{ij} \boldsymbol{x}_i \boldsymbol{x}_j}{\sum_{x_i > 0} \boldsymbol{d}_i} + \frac{\sum_{(x_i < 0, x_j > 0)} -w_{ij} x_i x_j}{\sum_{x_i < 0} d_i} \tag{786}$$

Let $\mathbf{D}$ be an $N \times N$ diagonal matrix with $\mathbf{d}$ on its diagonal and $\mathbf{W}$ be an $N \times N$ symmetric matrix with $W(i, j) = w_{ij}$,

$$k = \frac{\sum_{x_i > 0} \boldsymbol{d}_i}{\sum_i \boldsymbol{d}_i} \tag{787}$$

Denote an $N \times 1$ of all ones as $\mathbf{1}$. It is easy to see that $(1 + \mathbf{x})/2$ and $(1 - \mathbf{x})/2$ are indicator vectors for $x_i > 0$ and $x_i < 0$, respectively, we can rewrite $4[\text{Ncut}(\mathbf{x})]$ as:

$$\begin{aligned} 4[\text{Ncut}(\mathbf{x})] &= \frac{(\mathbf{1}+\boldsymbol{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+x)}{k\mathbf{1}^T\mathbf{D}1} + \frac{(\mathbf{1}-x)^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-x)}{(1-k)\mathbf{1}^T\mathbf{D}1} \\ &= \frac{\left(\boldsymbol{x}^T(\mathbf{D}-\mathbf{W})x + \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}\right)}{k(1-k)\mathbf{1}^T\mathbf{D}1} + \frac{2(1-2k)\mathbf{1}^T(\mathbf{D}-\mathbf{W})x}{k(1-k)\mathbf{1}^T\mathbf{D}\mathbf{1}} \end{aligned} \tag{788}$$

Let

$$\alpha(\mathbf{x}) = \mathbf{x}^T(\mathbf{D}-\mathbf{W})\mathbf{x} \tag{789}$$

$$\beta(\mathbf{x}) = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{x} \tag{790}$$

$$\gamma = \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1} \tag{791}$$

and

$$M = \mathbf{1}^T\mathbf{D}\mathbf{1} \tag{792}$$

We can expand simplify our expression for $4[\text{Ncut}(\mathbf{x})]$ as

$$\begin{aligned} 4[\text{Ncut}(\mathbf{x})] &= \frac{(\alpha(x)+\gamma) + 2(1-2k)\beta(x)}{k(1-k)M} = \frac{(\alpha(x)+\gamma) + 2(1-2k)\beta(x)}{k(1-k)M} - \frac{2(\alpha(x)+\gamma)}{M} + \frac{2\alpha(x)}{M} + \frac{2\gamma}{M} \\ &= \frac{\left(1-2k+2k^2\right)(\alpha(x)+\gamma) + 2(1-2k)\beta(x)}{k(1-k)M} + \frac{2\alpha(x)}{M} \\ &= \frac{\frac{\left(1-2k+2k^2\right)}{(1-k)^2}(\alpha(\boldsymbol{x})+\gamma) + \frac{2(1-2k)}{(1-k)^2}\beta(\boldsymbol{x})}{\frac{k}{1-k}M} + \frac{2\alpha(x)}{M} \end{aligned} \tag{793}$$

Let $b = (1-k)/k$ and since $\gamma = 0$, we can simplify this expression as

$$\begin{aligned} 4[\text{Ncut}(\mathbf{x})] &= \frac{\left(1+b^2\right)(\alpha(x)+\gamma) + 2\left(1-b^2\right)\beta(x)}{bM} + \frac{2b\alpha(x)}{bM} \\ &= \frac{\left(1+b^2\right)(\alpha(x)+\gamma)}{bM} + \frac{2\left(1-b^2\right)\beta(x)}{bM} + \frac{2b\alpha(x)}{bM} - \frac{2b\gamma}{bM} \\ &= \frac{\left(1+b^2\right)\left(x^T(\mathbf{D}-\mathbf{W})x + \mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}\right)}{b\mathbf{1}^T\mathbf{D}1} + \frac{2\left(1-b^2\right)\mathbf{1}^T(\mathbf{D}-\mathbf{W})x}{b\mathbf{1}^T\mathbf{D}1} + \frac{2bx^T(\mathbf{D}-\mathbf{W})x}{b\mathbf{1}^T\mathbf{D}1} - \frac{2b\mathbf{1}^T(\mathbf{D}-\mathbf{W})\mathbf{1}}{b\mathbf{1}^T\mathbf{D}1} \\ &= \frac{(1+x)^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+\boldsymbol{x})}{b\mathbf{1}^T\mathbf{D}\mathbf{1}} + \frac{b^2(\mathbf{1}-x)^T(\mathbf{D}-\mathbf{W})(\mathbf{1}-x)}{b\mathbf{1}^T\mathbf{D}1} - \frac{2b(\mathbf{1}-x)^T(\mathbf{D}-\mathbf{W})(\mathbf{1}+x)}{b\mathbf{1}^T\mathbf{D}1} \\ &= \frac{[(\mathbf{1}+x) - b(\mathbf{1}-x)]^T(\mathbf{D}-\mathbf{W})[(\mathbf{1}+x) - b(\mathbf{1}-x)]}{b\mathbf{1}^T\mathbf{D}1} \end{aligned} \tag{794}$$

Let $\mathbf{y} = (\mathbf{1}+\mathbf{x}) - b(\mathbf{1}-\mathbf{x})$, then we can see that

$$y^T\mathbf{D}1 = \sum_{x_i>0} d_i - b\sum_{x_i<0} d_i = 0 \tag{795}$$

since $b = k/(1-k) = \sum_{x_i>0}\mathbf{d}_i / \sum_{x_i<0}\mathbf{d}_i$ then

$$\boldsymbol{y}^T\mathbf{D}\boldsymbol{y} = \sum_{x_i>0}\boldsymbol{d}_i + b^2\sum_{x_i<0}\boldsymbol{d}_i = b\sum_{x_i<0} d_i + b^2\sum_{x_i<0} d_i == b\left(\sum_{x_i<0}\boldsymbol{d}_i + b\sum_{x_i<0}\boldsymbol{d}_i\right) = b\mathbf{1}^T\mathbf{D}\mathbf{1} \tag{796}$$

Thus,

$$\min_x Ncut(x) = \min_y \frac{y^T(\mathbf{D}-\mathbf{W})y}{y^T\mathbf{D}y} \tag{797}$$

with the condition $\mathbf{y}(i) \in \{1, -b\}$ and $\mathbf{y}^T\mathbf{D1} = 0$. It turns out that this is a classical eigenvalue problem. We can adjust y to take any real value. Our eigenvalue problem, in general is

$$(\mathbf{D} - \mathbf{W})\boldsymbol{y} = \lambda \mathbf{D} y \tag{798}$$

With a positive semi-definite matrix $\mathbf{D}$, then we can rewrite

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}} z = \lambda z \tag{799}$$

Now let us consider a three node problem. Let us consider there are 3 nodes such that 2 nodes are in group A and 1 node is in group B. So, $A \quad A \quad B$ corresponds to $x = [1 \quad 1 \quad -1]$. Then, we can write

$$\sum_{x_i>0, x_j<0} -w_{ij} x_i x_j \tag{800}$$

Based on this, the only non-negative values are

$$-w_{13} x_1 x_3 = w_{13} \tag{801}$$
$$-w_{23} x_2 x_3 = w_{13} \tag{802}$$

Expanding the first numerator of the normalized cut

$$(1+x)^T(D-w)(1+x) = \begin{bmatrix} 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} w_{12}+w_{13} & -w_{12} & -w_{13} \\ -w_{21} & w_{21}+w_{23} & -w_{23} \\ -w_{31} & -w_{32} & w_{31}+w_{32} \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} = 4[w_{13}+w_{23}] \tag{803}$$

Similarly, expanding out k,

$$k = \frac{\sum_{x_i>0} d_i}{\sum_i d_i} = \frac{w_{11}+w_{21}+w_{22}+w_{13}+w_{23}+w_{12}}{w_{11}+w_{12}+w_{13}+w_{21}+w_{22}+w_{23}+w_{31}+w_{32}+w_{33}} \tag{804}$$

and expanding out the denominator for the normalized cut from that same term

$$\begin{aligned} k\mathbf{1}^T D\mathbf{1} &= \frac{w_{11}+w_{21}+w_{22}+w_{13}+w_{23}+w_{12}}{w_{11}+w_{12}+w_{13}+w_{21}+w_{22}+w_{23}+w_{31}+w_{32}+w_{33}} \times \ldots \\ &\ldots (w_{11}+w_{12}+w_{13}+w_{21}+w_{22}+w_{23}+w_{31}+w_{32}+w_{33}) \\ &= w_{11}+w_{21}+w_{22}+w_{13}+w_{23}+w_{12} = \sum_{x_i>0} d_i \end{aligned} \tag{805}$$

From the normalized cut derivation

$$y^T\mathbf{D}1 = \sum_{x_i>0} d_i - b \sum_{x_i<0} d_i = 0 \tag{806}$$

For this equation, note that

$$b = \frac{k}{1-k} = \frac{w_{11}+w_{12}+w_{21}+w_{22}+w_{13}+w_{23}}{w_{31}+w_{32}+w_{33}} \tag{807}$$
$$\sum_{x_i>0} d_i = w_{11}+w_{12}+w_{21}+w_{22}+w_{13}+w_{23} \tag{808}$$
$$\sum_{x_i<0} d_i = w_{31}+w_{32}+w_{33} \tag{809}$$

Thus

$$y^T\mathbf{D}1 = w_{11}+w_{12}+w_{13}+w_{22}+w_{23}+w_{21} - b\,(w_{31}+w_{32}+w_{33}) = 0 \tag{810}$$

From here, we solve the generalized eigenvalue problem

$$(D-w)y = \lambda Dy[\text{ generalized eigen eystem }] \tag{811}$$

$$\begin{gathered}(D-w)D^{-\frac{1}{2}}z = \lambda DD^{-\frac{1}{2}}z\left(\text{ since } y = D^{-\frac{1}{2}}z\right) \\ D^{-\frac{1}{2}}(D-w)D^{-\frac{1}{2}}z = \lambda D^{-\frac{1}{2}}DD^{-\frac{1}{2}}z\end{gathered} \tag{812}$$

$$D^{-\frac{1}{2}}(D-w)D^{-\frac{1}{2}}z = \lambda z[\text{ standard eigen system }] \tag{813}$$

Note that is only true if $\mathbf{D}$ is positive semi definite. In the literature, $\mathbf{W}-\mathbf{D}$ is known as the Laplacian matrix. The smallest eigenvalue, as $\mathbf{D}$ is PSD, must be 0. So, for this scheme, we consider the second smallest eigenvalue and the corresponding eigenvector $z_1 = y_1$. The positive elements in y_i corresponding to partition A, while the negative elements correspond to partition B, for the 2 node case. To construct the features for which we perform the normalized cut,

$$w_{ij} = e^{\frac{-\|\boldsymbol{F}_{(i)}-\boldsymbol{F}_{(j)}\|_2^2}{\sigma_I^2}} = \begin{cases} \frac{-\left\|x_{(i)}-x_{(j)}\right\|_2^2}{\sigma_X^2} & \text{if } \|X(i)-X(j)\|_2 < r \\ 0 & \text{otherwise}\end{cases} \tag{814}$$

$\mathbf{x}(i)$ can be any feature, including intensity, color, filter outputs, scales, or a Gabor filter. This can be done by means of MRFs but to do this, we need to fit an MRF, which requires training. However, the normalized cut approach has the advantage of not requiring training.