Computer Vision

MSc Informatics option GVR

James L. Crowley

Fall Semester 24 November 2016

Lesson 6

Corner Detectors, Ridge Detection, and the Viola Jones Face Detector

Lesson Outline:

Notation

$\{W_m\}$	Training set of M windows (imagettes) for learning.
$\{y_m\}$	Indicator variable for each trainig window. $y_m \in \{0,1\}$ $y_m=1$ if the window contains a face and otherwise $y_m=0$
M	The number of training samples.
$X_n = \sum_{x=1}^{C} \sum_{y=1}^{R} W(x,y) H_n(x,y)$	A Haar-like feature applied to the window (imagette) W
$X_n = \langle W, H_n \rangle$	A Haar-like feature is an inner product with a receptive field
$h_n(W) = h_n(X_n) = \begin{cases} 1 & \text{if } p_n(X_n + b_n) > 0 \\ 0 & otherwise \end{cases}$	a weak classifier (hypothesis) for the feature X_n.
p_n	Polarity (Sign) of the for the classifier $h_n(W)$ $p_n \in \{-1,1\}$
b_n	Constant for the classifier $h_n(W)$. Acts as a threshold for $p_n X_n$
$h(W) = \sum_{t=1}^{T} \alpha_t h_t(W)$	A weighted Committee of weak classifiers
$E_T = \frac{1}{M} \sum_{m=1}^{M} w_m \lvert h(W_m) - y_m \rvert$	The error rate for the committee for a data set $\{W_m\}$

1. Harris Corner Detector

Harris, Chris, and Mike Stephens. "A combined corner and edge detector." Alvey vision conference. Vol. 15. 1988.

The Harris-Stevens Corner detector is inspired from the Moravec Interest Point detector proposed in 1973 by Hans Moravec for stereo matching. Moravec used the Sum of Squared Difference (SSD) between adjacent small patches to detect interest points. In 1988, Harris and Stevens observed that this is equivalent to an auto-correlation of the image.

$$S(x,y) = \sum_{u,v} w(u,v)\big(I(u+x,v+y) - I(u,v)\big)^2$$

where $I(x,y)$ is the image,
w(x,y) is some window function, typically Gaussian.

I(u+x,v+y) can be approximated as a local Taylor Series:

$$I(u+x,v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y$$

where $I_x(x,y)$ and $I_y(x,y)$ are the local x and y derivatives

Giving $S(x,y) = \sum_{u,v} w(u,v)\big(I_x(u,v)x + I_y(u,v)y\big)^2$

Which can be written in Matrix form as: $S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix}$ where A is the "Structure Tensor"

$$A = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

With our Gaussian pyramid this is simply: $A = \begin{bmatrix} P_x^2 & P_x P_y \\ P_x P_y & P_y^2 \end{bmatrix}$

Compute the Eigenvectors of A: $\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} = R\,A\,R^T$

where λ_1 is the maximum gradient, λ_2 is the minimum gradient.

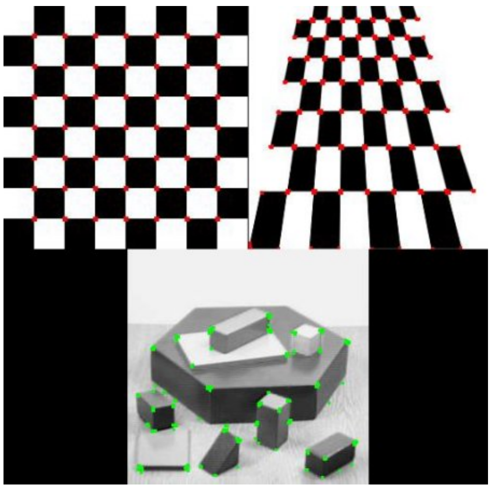
if $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ then the point is of no interest
if $\lambda_1 \approx 0$ and $\lambda_2 >> 0$ then the point is a horizontal edge
if $\lambda_1 >> 0$ and $\lambda_2 \approx 0$ then the point is a vertical edge
if $\lambda_1 \approx \lambda_2 >> 0$ then the point is corner

To avoid computing the eigenvalues (requires a square root), we can define a measure for "corner-ness":

$$M_c = \det(A) - \kappa \cdot Trace^2(A) = \lambda_1\lambda_2 - \kappa\left(\lambda_1 + \lambda_2\right)^2$$

where κ is a tunable sensitivity parameter.

Examples of Harris-Stevens Corners:



2. Ridge Detection.

The Eigenvalues of the Hessian provide a popular ridge detector.

The Hessian at scale s is $H(x,y,s) = \begin{pmatrix} P_{xx}(x,y,s) & P_{xy}(x,y,s) \\ P_{xy}(x,y,s) & P_{yy}(x,y,s) \end{pmatrix}$

The Eigenvalues are found by diagonalizing the Hessian.
For any point in scale space *(x, y, s)*

$$\begin{pmatrix} P_{rr} & 0 \\ 0 & P_{ss} \end{pmatrix} = R\,H\,R^T \qquad \text{where} \qquad R = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

P_{ss} is the largest value in second derivative, while P_{rr} is the smallest.
On a ridge point, P_{rr} will be the second derivative along the ridge (close to zero) while P_{ss} will be the 2nd derivative perpendicular to the ridge.

For any 2D Matrix, the principal directions can be computed directly as

$$\cos(\theta) = \sqrt{\frac{1}{2}\left(1 + \frac{P_{xx} - P_{yy}}{\sqrt{\left(P_{xx} - P_{yy}\right)^2 + 4P_{xy}^2}}\right)}, \quad \sin(\theta) = \operatorname{sgn}(P_{xy})\sqrt{\frac{1}{2}\left(1 - \frac{P_{xx} - P_{yy}}{\sqrt{\left(P_{xx} - P_{yy}\right)^2 + 4P_{xy}^2}}\right)}$$

Recall that the gradient is $\vec{\nabla}P(x,y,s) = \begin{pmatrix} P_x(x,y,s) \\ P_y(x,y,s) \end{pmatrix} = \begin{pmatrix} P * G_x(x,y,s) \\ P * G_y(x,y,s) \end{pmatrix}$

for any point *(x, y, s)*, the Gradient can be aligned with the ridge using

$$P_r = \cos(\theta)P_x - \sin(\theta)P_y$$
$$P_s = \sin(\theta)P_x + \cos(\theta)P_y$$

A positive ridge point is any point, *R(x,y,s)* that satisfies:

$P_r = 0$ and $P_{rr} \le 0$ and $|P_{rr}| \ge |P_{ss}|$

A negative ridge is any point for which

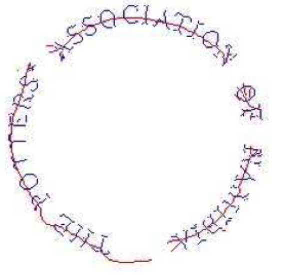
$P_r = 0$ and $P_{rr} \ge 0$ and $|P_{rr}| \le |P_{ss}|$

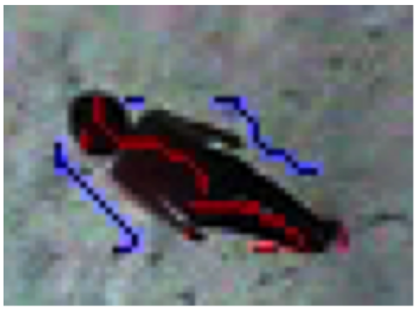
of course, P_r will rarely be exactly zero, so we use form of approximation $P_r \approx 0$

The ridge direction at (x, y, s) is: $\cos(\theta) = \frac{P_x}{\sqrt{P_x^2 + P_y^2}}$ $\sin(\theta) = \frac{P_y}{\sqrt{P_x^2 + P_y^2}}$

A Maximal ridge is a ridge point $R(x,y,s)$ for which $\underset{s}{\text{local}-\max}\{\nabla^2 P(x,y,s)\}$

Examples of Maximal Ridge points:





3. Pattern Detectors

Our problem is to build a function, called a classifier or recognizer, $\mathrm{R}(\vec{X})$, that maps the observation, $\vec{X}$ into a statement that the observation belongs to a class $\hat{C}_k$ from a set of K possible classes. $R(\vec{X}) \rightarrow \hat{C}_k$

$$\vec{X} \rightarrow \boxed{R(\vec{X})} \rightarrow \hat{C}_k$$

$\vec{X}$ is a feature vector composed of *D* properties or features $\qquad \vec{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$

$\vec{X}$ can represent the pixels in an image window, W(i,j).

$\vec{X}$ can be a "feature" vector for appearance, $\vec{A}(i,j)$, in an image neighborhood obtained by projection of an image neighborhood *P(i,j)* onto this set of functions,

$$\vec{A}(i,j) = \begin{pmatrix} a_1 \\ a_2 \\ \cdots \\ a_K \end{pmatrix} \text{ where } a_k(i,j) = \sum_{x=-R}^{R} \sum_{y=-R}^{R} p(i-x, j-y) G_k^{\sigma}(x,y)$$

$\vec{X}$ can even be the entire image.

The class $\hat{C}_k$ is from a set of K known classes $\{C_k\}$.

The classes can be local structures (pre-attentive features) such as spots, corners, bar-ends and bars.
The classes can also be categories of objects such as faces, pedestrians or bicycles.

Almost all current classification techniques require the set of classes, $\{C_k\}$, to be predefined. An interesting open research problem is how to design classification algorithms that allow $\{C_k\}$ to be an open set that grows with experience.

Pattern detectors are a special case where K=2.
In this case, C_1, is the target class and C_2 is everything else.

3.1. Discriminant and Decision Functions

The classification function $R(\vec{X})$ can typically be decomposed into two parts:

$$\hat{C}_k \leftarrow d\left(\vec{g}\left(\vec{X}\right)\right)$$

where $\vec{g}\left(\vec{X}\right)$ is a discrminant function and $d\left(\vec{g}\left(\vec{X}\right)\right)$ is a decision function.

$\vec{g}\left(\vec{X}\right)$: A discriminant function that transforms: $\vec{X} \rightarrow \mathbb{R}^K$
(A vector of real numbers)

$d\left(\vec{g}\left(\vec{X}\right)\right)$: A decision function $\mathbb{R}^K \rightarrow \hat{C}_k \in \{C_k\}$

The discriminant is typically a vector of functions, with one for each class.

$$\vec{g}(\vec{X}) = \begin{pmatrix} g_1(\vec{X}) \\ g_2(\vec{X}) \\ \vdots \\ g_K(\vec{X}) \end{pmatrix}$$

The decision function, $d\left(\vec{g}\left(\vec{X}\right)\right)$, can be an arg-max{}, a sigma function, a logistics function, or any other function that selects C_k from $\vec{X}$.
For today we will use arg-max{}.

$$\hat{C}_k = d\left(\vec{g}\left(\vec{X}\right)\right) = \arg\text{-}\max_{C_k}\left\{g_k\left(\vec{X}\right)\right\}$$

In some problems, there is a notion of "cost" for errors that the cost is different for different decisions. In this case we will seek to minimize the cost of an error rather than the number of errors by biasing the classification with a notion of risk.

For a two class problem it is possible to use a single discriminant, $g(\vec{X})$

The detection function has the form: $R(\vec{X}) = d(g(\vec{X}))$

3.2. ROC Curves

Two-class classifiers have long been used for signal detection problems in communications and have been used to demonstrate optimality for signal detection methods. The quality metric that is used is the Receiver Operating Characteristic (ROC) curve. This curve can be used to describe or compare any method for signal or pattern detection.

The ROC curve is generated by adding a variable Bias term to a discriminant function.

$$R(\vec{X}) = d(g(\vec{X}) + B)$$

and plotting the rate of true positive detection vs false positive detection where $R(\vec{X}_m)$ is the classifier as in lesson 1. As the bias term, B, is swept through a range of values, it changes the ratio of true positive detection to false positives.

For a ratio of histograms, $g(\vec{X}_m)$ is a probability ranging from 0 to 1.
B can range from less than *–0.5* to more than *+0.5*.
When $B \leq -0.5$ all detections will be Negative.
When $B > +0.5$ all detections will be Positive.
Between *–0.5* and *+0.5* $R(\vec{X})$ will give a mix of TP, TN, FP and FN.

The bias term, B, can act as an adjustable gain that sets the sensitivity of the detector.
The bias term allows us to trade False Positives for False Negatives.

The resulting curve is called a Receiver Operating Characteristics (ROC) curve.
The ROC plots True Positive Rate (TPR) against False Positive Rate (FNR) as a function of B for the training data $\{\vec{X}_m\}$, $\{y_m\}$.

For each training sample, the detection as either Positive (P) or Negative (N)

IF $g(\vec{X}_m)+B > 0.5$ THEN P else N

The detection can be TRUE (T) or FALSE (F) depending on the indicator variable y_m

IF $y_m = R(\vec{X}_m)$ THEN T else F

Combining these two values, any detection can be a True Positive (TP), False Positive (FP), True Negative (TN) or False Negative (FN).

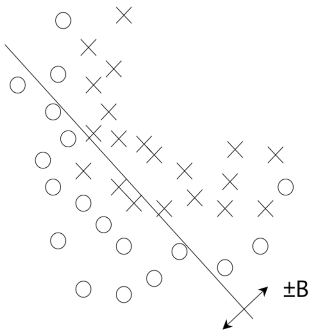
For the M samples of the training data $\{\vec{X}_m\}$, $\{y_m\}$ we can define:
 #P as the number of Positives,
 #N as the number of Negatives,
 #T as the number of True and
 #F as the number of False,
From this we can define:
 #TP as the number of True Positives,
 #FP as the number of False Positives,
 #TN as the number of True Negative,
 #FN as the number of False Negatives.

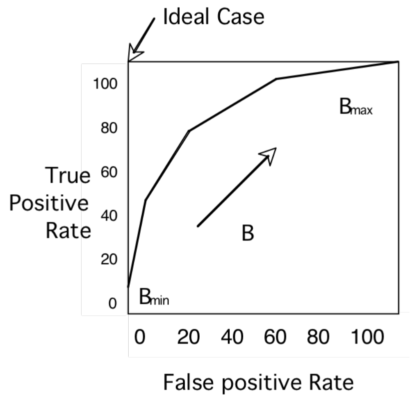
Note that #P = #TP + #FN
And #N = #FP+ #TN

The True Positive Rate (TPR) is $TPR = \frac{\#TP}{\#P} = \frac{\#TP}{\#TP + \#FN}$

The False Positive Rate (FPR) is $FPR = \frac{\#FP}{\#N} = \frac{\#FP}{\#FP + \#TN}$

The ROC plots the TPR against the FPR as a bias B is swept through a range of values.





When B is less than –0.5, all the samples are detected as N, and both the TPR and FPR are 0. As B increases both the TPR and FPR increase. Normally TPR should rise monotonically with FPR. If TPR and FPR are equal, then the detector is no better than chance.

The closer the curve approaches the upper left corner, the better the detector.

		$y_m = R(\vec{X}_m)$	
		T	F
$d(g(\vec{X}_m) + B > 0.5)$	P	True Positive (TP)	False Positive (FP)
	N	False Negative (FN)	True Negative (TN)

4. The Viola Jones Face Detector

In 2001, Paul Viola and Mike Jones at MERL (Misubishi Research Labs) demonstrated a revolutionary new technique to detect faces in images. Their technique used a very large number of very simple features using difference of boxes. They used a technique called boosted learning to learn committees of simple classifiers using difference of boxes. They applied this technique with a brute force "scanning window" approach in which rectangular windows of a given size are independently classified as "Face" or "Not Face".

Each window was described with a set of features computed from Difference of Boxes. Box features are sums of pixels over rectangular boxes and can be computed with a very fast algorithm known as "integral images". They referred to these as "Haar-like" features because of the Haar Transform, a form of binary Discrete Fourier transform used in signal processing. This gave a VERY large number of potential features.

Each difference of Box feature defines a linear classifier for whether the window contained a face. Some classifiers are better than others. They used boosted learning to construct a committee of linear classifiers composed of the most effective at detecting face. Each classifier (Haar-like feature) produced a yes/no vote as to whether the window contained a face. The sum of the votes determined whether the committee decided Face or Not face.

They learned a sequence of committees for face detection, where each committee was trained on only those windows that were selected as "face" by the previous committee.

This process took around 3 months to train, but once trained provided revolutionary gains in the ability to detect faces in images. The learned process was patented by MERL, and published in OpenCV, allowing the community to verify the effectiveness of the process and to use it for proto-typing applications.

Scanning Window Pattern Detectors.

A scanning window is a brute force method to test if a pattern can be found in an image.

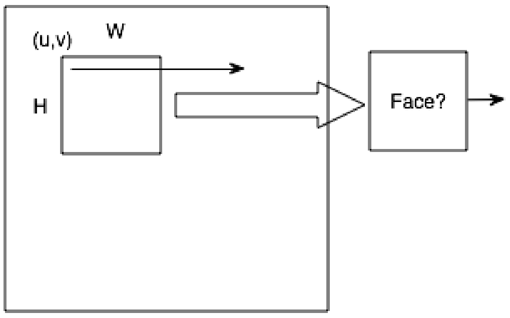
Assume "gray-scale" image, *p(i,j)*, in which each pixel is an 8 bit luminance value.

An image window, or "imagette" is a rectangular region of the image. A window can be defined by two points: the top-left and bottom-right corners. This may be represented by a vector (t, l, b, r). Note that the origin is the upper left corner. Columns are numbered x=1 to W and rows are numbered y=1 to H

For any pixel (*i, j),* we can define a window, *W(x,y),* of size C columns by R rows using the pixels from *p(i, j)* to *p(i+C-1, j+R-1)*.

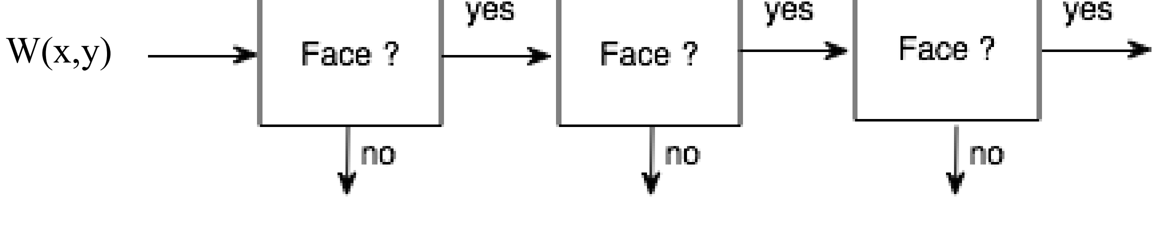
$$W(x,y) = p(i+x-1, j+y-1) \quad \text{for x from 1 to C and y from 1 to R.}$$

For faces detection, the window is typically from 16 x 16 to 32 x 32 pixels. The Viola Jones face detector was trained with windows of size (24, 24) pixels.



Note that it is possible to use an affine transform to map a region of the image of any size into the standard sized window, W(x,y). This is called a texture map.

The decision of whether the window *W(x,y)* contains a face is provided by a cascade of boosted linear classifiers.



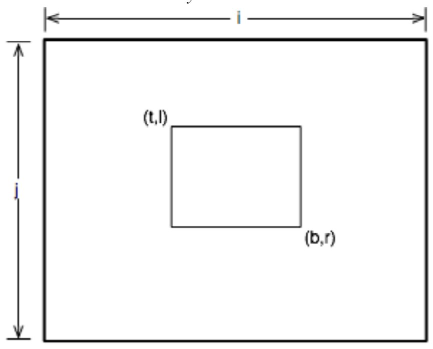
The algorithm requires a large number of local "features" to classify the window.

5. Image Description with Difference of Box Features

Box Features

A box feature is the sum of pixels within a rectangle
Assume a rectangle from top (t) and left (l) to bottom (b) and right (r), with the constraints: top < bottom and right > left.

$$b(t,l,b,r) = \sum_{x=l}^{r}\sum_{y=t}^{b} W(x,y)$$



Difference of boxes can be used as features for classification.
Viola-Jones uses three kinds of Difference of Box features:

1) Two-rectangle features, computed as differences of two adjacent rectangular boxes of the same size. The boxes are the same size and shape and are adjacently aligned so that they share one side.

2) Three rectangle features, computed as the sum of two outside rectangles subtracted from an internal center rectangle.

3) Four rectangle features, computed as the difference of diagonal pairs of rectangles.

For a window of 24 x 24 pixels, this gives more than 180,000 possible features!
(136, 336 features according to Wikipedia).

Computation is very fast because of the use of integral images.

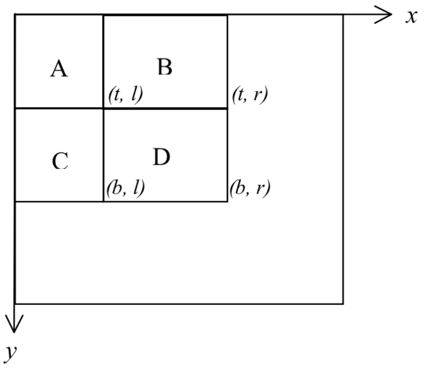
Integral Images

An integral image is an image where each pixel contains the sum from the upper left corner. We can build an integral image from our image window *W(x, y)* with:

$$ii(x,y) = \sum_{i=1}^{x}\sum_{j=1}^{y} W(i,j)$$

Each sample in the integral image represents the sum of pixels from the upper left corner.

An integral image provides a structure for very fast computation of box features. Note that a sum of pixels in a rectangle can be computed from an integral image using only 4 operations (additions/subtractions).



Consider four adjacent rectangular regions A, B, C, D.

Note that *ii(t, l) = A.* *ii (t,r) = A+B* *ii(b,l)=A+C* *ii(b,r)=A+B+C+D*

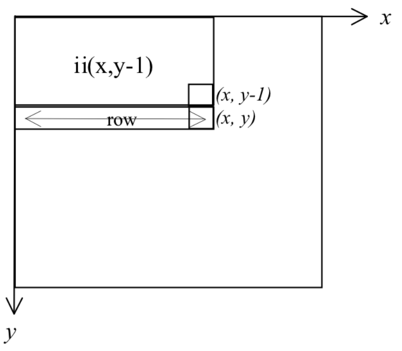
The box: box(t, l, b, r) is *A+B+C+D – (A+B) – (A+C) + A*

box(t, l, b, r) = A+B+C+D – (A+B) – (A+C) + A
box(t, l, b, r) = ii(b, r) – ii(t, r) – ii(b, l) + ii(t, l)

Fast Integral Image algorithm.

Integral images have been used for decades to compute local energy for normalization of images. For an R x C window of an image, W(i,j), the integral image, ii(x,y), is computed with a recursive algorithm that uses an intermediate buffer, "row", for a running sum of pixels within the current row.

```
ii(1,1) = W(1, 1)
For x = 2 to C
        ii(x,1) = ii(x-1,1) + W( x, 1)
For y = 2 to R
{       row=0   // reset the row buffer //
        For x = 1 to C
        {       row = row +  W(x,y)
                ii(x,y) = ii(x,y-1) + row
        }
}
```



Note that many authors use a less efficient algorithm that requires computing running sums of all the columns.

Difference of Adjacent Boxes Features

A box feature is the sum of pixels in a rectangle. With integral images, a box feature costs 3 ops. (an add, subtract, or multiply is 1 op)

$$B(t, l, b, r) = ii(b,r) – ii(t,r) – ii(b,l) + ii(t,l)$$

Two rectangle features

A first order Difference of Boxes (DoB) feature is a difference of two boxes

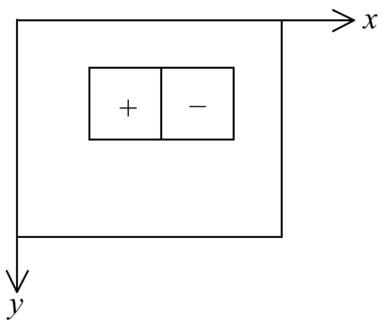
$$DoB(t_1,l_1,b_1,r_1,t_2,l_2,b_2,r_2) = box(t_1,l_1,b_1,r_1) – box(t_2,l_2,b_2,r_2)$$

An arbitrary 1st order difference of boxes costs 7 ops.

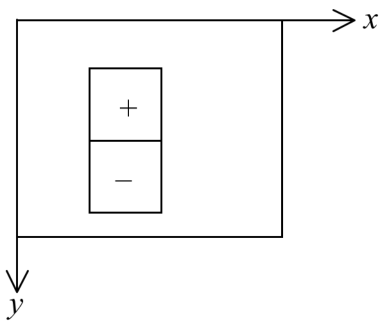
$$DoB(t_1,l_1,b_1,r_1,t_2,l_2,b_2,r_2) = box(t_1,l_1,b_1,r_1) – box(t_2,l_2,b_2,r_2)$$

$$= ii(b_1,r_1) – ii(t_1,r_1) – ii(b_1,l_1) + ii(t_1,l_1) – [\ ii(b_2,r_2) – ii(t_2,r_2) – ii(b_2,l_2) + ii(t_2,l_2)\ ]$$

Difference of Adjacent Boxes uses boxes of the same size that share a side.
There are two possible cases: Difference of Horizontally Adjacent Boxes (DoHAB) and Difference of Vertically Adjacent Boxes DoVAB



DoHAB()



DoVAB()

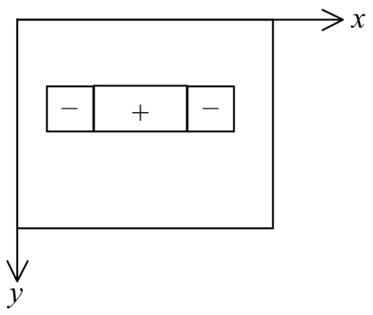
If the boxes share a vertical boundary, then $t_1 = t_2 = t$, $b_1 = b_2 = b$ and $r_1 = l_2$, and the boxes are horizontally adjacent.

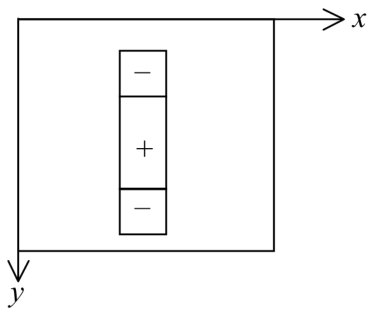
If the boxes share a horizontal boundary then $b_1 = t_2$ and the boxes are vertically adjacent and $l_1 = l_2$ and $r_1 = r_2$

The fact that both rectangles are the same size, guarantees that the feature is zero for a constant region. The difference of adjacent boxes costs 7 ops. (2 mults, 3 subtracts, 2 adds)

Three rectangle features

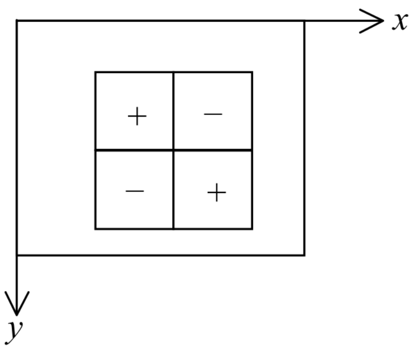
Three rectangle features are computed as the sum of two outside rectangles subtracted from an internal center rectangle. The size of the inner rectangle is twice the size of the outer rectangles. This guarantees that the sum is zero when covering a uniform region.





Three rectangle features cost 11 ops.

Four rectangle features



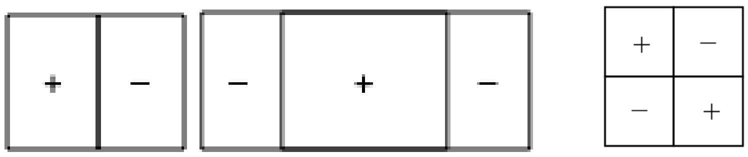
Four rectangle features, computed as the difference of diagonal pairs of rectangles. Difference of adjacent boxes are similar to Haar wavelets.

Note that a difference of boxes can be seen as a computing an inner product of the window with a filter *$H_n(x,y)$* (also called a mask, or a receptive field).

$$X_n = \sum_{x=1}^{C} \sum_{y=1}^{R} W(x,y)H_n(x,y)$$

6. Linear Classifiers for Face Detection

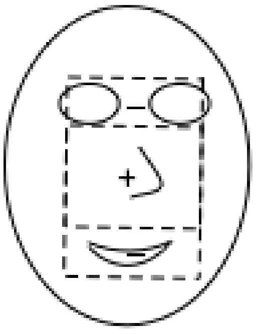
Let us assume a set of N image features, *$\{X_n\}$* computed from difference of adjacent boxes. For a 24x24 window, there will be over 180,000 two, three and four box features.



These can be seen as defining a 180,000 dimensional space for classifying imagettes.

Each feature, X_n is one of the 180,000 difference of box features computed from the window, W.

$$X_n = \sum_{x=1}^{C} \sum_{y=1}^{R} W(x,y) H_n(x,y)$$



Some features respond to the appearance of a face. These can be used to determine if the imagette contains a face or not.

Each image feature specifies a weak classifier for the window: $h_n(W)$

$$h_n(W) = \begin{cases} 1 & \text{if } p_n(X_n + b_n) > 0 \\ 0 & \textit{otherwise} \end{cases}$$

where p_n is a "polarity" of +1 or -1 and b_n is a bias. $h_n(W)$ represents a hypothesis. Each weak classifier $h_n(W)$ corresponds to a Difference of Box feature $H_n(x,y)$ applied to the window $W(x,y)$.

(note that in their paper, Viola-Jones use x for the window W, and f_n for the nth difference of box feature).

Each weak classifier, $h_n(W)$ can be seen as a hyper-plane that partitions the hyper-dimensional feature space of imagettes of size 24x24. The problem is to choose the best $h_n(W)$ so that most non-face windows are on one side of the hyper-plane and most face windows are on the other.

To do this we will use a "training" set of M windows, $\{W_m\}$.
Each training window is labeled with an "indicator variable" y_m.

For imagettes that contain faces, $y_m = 1$. Imagettes that do not contain faces, $y_m = 0$.

Training a committee of classifiers

Assume a set of *M* face windows $\{W_m\}$ that have been labeled by a set of labels $\{y_m\}$ such that y=+1 if face and y=0 if not face.

Then for any imagette, W_m, each feature "votes" for a face (P for positive) or not a face (N for negative).

$$h_n(W) = \begin{cases} 1 & \text{if } p_n(X_n + b_n) > 0 \\ 0 & \textit{otherwise} \end{cases}$$

For a training set of M windows, $\{W_m\}$, the detection rate (or positive rate) for a weak classifier $h_n(W)$ is the percentage of positive detections.

$$P_n = \frac{1}{M}\sum_{m=1}^{M} h_n(W_m)$$

Positive detections can be true positive and false positives.

Whether a detection is true (T) of false (F) can be determined by the indicator variable. $y_m=1$ if W_m contains a face, and $y_m=0$ otherwise.

if $| h_n(W_m) - y_m | = 1$ then FALSE else TRUE.

For the training set of *M* windows, $\{W_m\}$, the error rate for a weak classifier $h_n(W)$ is the percentage of true classifications.

$$E_n = \frac{1}{M}\sum_{m=1}^{M} |h_n(W_m) - y_m|$$

Note that the error rate is a number between 0 and 1.

The classifier $h_n(W)$ that minimizes the error rate is

$$h_n = \arg-\min_n\{\sum_{m=1}^{M} |h_n(W_m) - y_m|\}$$

7. AdaBoost

AdaBoost (adaptive Boosting) is a meta-algorithm for learning a 2-class detection functions. Adaboost builds a strong classifier from a large number of weak classifiers. The outputs of the weak classifiers are combined as a weighted sum of votes. The resulting strong committee can be made arbitrarily good by adding more weak classifiers.

Adaboost is particularly useful in problems with a large number of features or large numbers of possible weak classifiers.

7.1. The Boosted Classifier

The boosted classifier can be seen as a form of Committee that decides using weighted votes by a set of T weak classifiers $h_i(W)$.

A weighted committee has the form:

$$h(W) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(W) \geq \frac{1}{2}\sum_{t=1}^{T}\alpha_t \\ 0 & otherwise \end{cases}$$

where $h_t(W)$ is a weak classifier and α_t is a learned weight for each weak classifier that depends on the error rate E_t

where $\alpha_t = \log\frac{1}{\beta_t}$ and $\beta_t = \frac{E_t}{1-E_t}$

Viola and Jones detector used AdaBoost to learn a committees of weak classifiers for faces in 24 x 24 pixel windows.

We assume a set of N weak classifier $h_n(W)$ maps a window W into a vote $v_i \in \{0,1\}$ using a difference of Box (Haar) feature $H_n(W)$.

$$h_n(W) = \begin{cases} 1 & \text{if } p_n(X_n + b_n) > 0 \\ 0 & otherwise \end{cases}$$

where: $X_n = \sum_{x=1}^{W}\sum_{y=1}^{H} W(x,y)H_n(x,y)$

For a 24 x 24 window, N = 136, 336 classifiers (according to wikipedia).

Boosted learning is an iterative procedure to choose weak classifiers and weights from a training set of M training windows, $\{W_m\}$ with their indicator variables $\{y_m\}$. Let #P be the number of positive training samples, and #N be the number of negative training samples.

The algorithm estimates a weight for each training sample, w_m.
The weights are initially set to

$$w_m = \begin{cases} \frac{1}{2\#P} & \text{if } y_m = 1 \\ \frac{1}{2\#N} & \text{if } y_m = 0 \end{cases}$$

(# is the cardinality operator - it counts the number of times something happens). Many authors assume that #P=#N and simply normalize to 1/M. This is not really valid. The algorithm then iterates over the number of weak classifiers.

The Algorithm
Initialize the algorithm with the weak classifier

$$h_1 = \arg\text{-}\min_{h_n}\{\sum_{m=1}^{M} w_m |h_n(W_m) - y_m|\}$$

the weight, α_1, is determined from the error rate: $E = \frac{1}{M}\sum_{m=1}^{M} w_m |h_1(W_m) - y_m|$

$$\alpha_1 = \log\left(\frac{1-E_T}{E_T}\right) \qquad \text{where } \beta_T = \frac{E_T}{1-E_T}$$

This is classifier t=1. Set T=1. Remove this from the set of available classifiers.

Initialize the weights as: $w_m = \begin{cases} \frac{1}{2\#P} & \text{if } y_m = 1 \\ \frac{1}{2\#N} & \text{if } y_m = 0 \end{cases}$

Loop: Let T=T+1

1) Normalize the weights to sum to 1. This converts w_m to a probability

$$S = \sum_{m=1}^{M} w_m \qquad ; \qquad w_m = \frac{w_m}{S}$$

2) For each Difference of Box feature, n, determining the polarity p_n and the threshold b_n that gives the best error rate with the current weights.

$$p_n, b_n = \arg-\max_{p,b}\left\{\sum_{m=1}^{M} w_m \left|h_n(W_m) - y_m\right|\right\}$$

This gives a new weak classifier $h_n(W) = \begin{cases} 1 & \text{if } p_n(X_n + b_n) > 0 \\ 0 & otherwise \end{cases}$

with error rate $E_n = \frac{1}{M}\sum_{m=1}^{M} w_m \left|h_n(W_m) - y_m\right|$

In this step the weights will bias the vote to give more strength to training samples that are improperly classified by the committee.

3) Choose the new weak classifier with the lowest error rate using the current weights. This is the T^th weak classifier is h_t, a_t:

$$h_T = \arg-\max_{h_n} \sum_{m=1}^{M} w_m \left|h_n(W_m) - y_m\right|$$

with the coefficient $\alpha_t = \log\frac{1-E_T}{E_T}$ where $E_T = \frac{1}{M}\sum_{m=1}^{M} w\left|h(W_m) - y_m\right|$

4) Use the error rate to update the weights to give more strength to windows that are in error. For each training sample, each weight w_m is multiplied by a factor β_m

$$w_m = w_m \beta_m$$

where $\beta_m = \begin{cases} \frac{E}{1-E} & \text{if } \sum_{t=1}^{T} \alpha_t h_t(W_m) - y_m < 0 \quad \text{FALSE} \\ 1 & otherwise \quad \text{TRUE} \end{cases}$

Loop until E_t below a specified error rate.

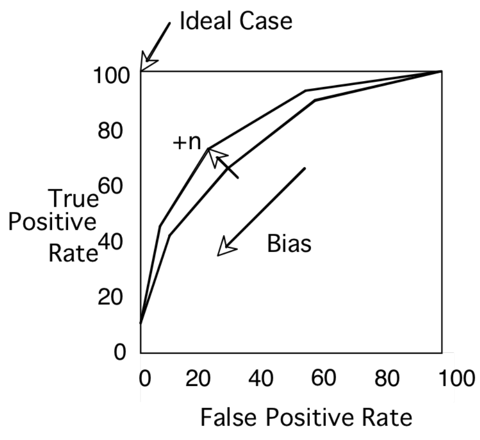
The final strong classifier is

$$h(W_m) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(W_m) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}$$

The new error rate for the committee is $E_T = \frac{1}{M}\sum_{m=1}^{M} w_m \left|h(W_m) - y_m\right|$

7.2. ROC Curve for a weighted committee

The ROC plots the True Positive Rate (TPR) against False Positive Rate (FPR) for a classifier as a function of the global bias B.



The Boosting theorem states that adding a each new weak classifier to a committee always improves the committee's ROC curve. We can continue adding classifiers until we obtain a desired rate of false positives and false negatives.

However, in general, the improvement provided for each new classifier becomes progressively smaller. We can end up with a very very large number of classifiers.

The halting criteria for boosted learning is set in terms of the FPR and TPR. When the ROC curve goes above for point (FPR, TPR) for some Bias B, the algorithm halts.

Note that the probability of error for a committee of classifiers can be computed for the training set as:

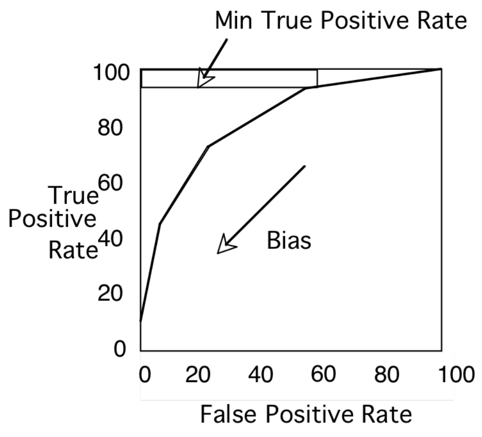
$$P(Error) = E_T = \frac{\# F}{M} = \frac{\# FP + \# FN}{M}$$

Where M is the number of training samples, and #F is the number of False detections (errors) within the M training samples.

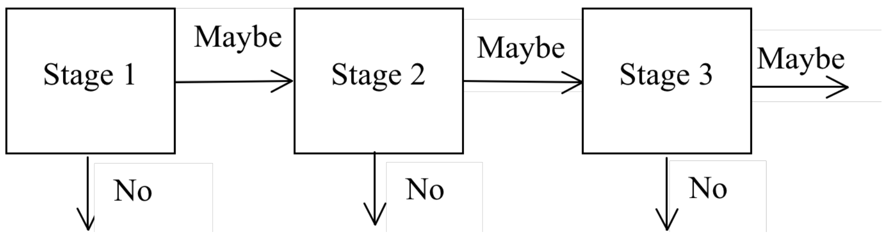
7.3. Learning a multi-stage cascade of classifiers

We can optimize the computation time by separating the committee into a multi-stage cascade of committees.

Each stage is composed of a committee that is designed with avoids rejecting possible true positives (high TPR: True Positive Rate) at the cost of accepting many False Positives (high False Positive Rate).



We construct each stage using only training data that passed the previous stage. Later stages are more expensive but are used less often.

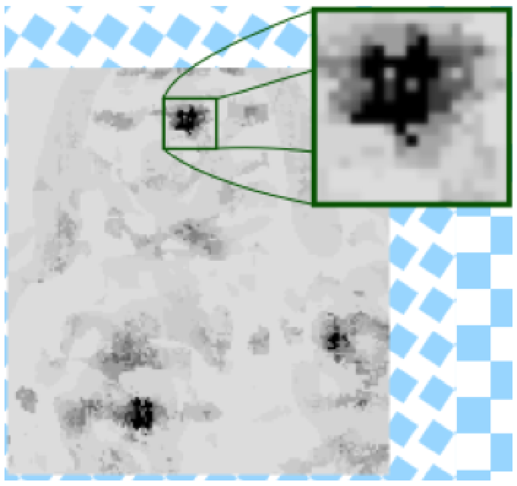


For each stage we set a minimum acceptable target for True Positives using the training data and accept the false positive rate that results.

Note that this can result in over-fitting the training data. It is important that the training data represent as large a variety of data as possible.

Each stage acts as a filter, rejecting a grand number of easy cases, and passing the hard cases to the next stage.

This is called a "cascade classifier"
Note that applying this to every position gives an "image" of cascade depths.



Faces can be detected as the center of gravity of "deep" detections.
Faces can be tracked using the Bayesian tracking described in the previous session.

This algorithm is part of the OpenCV. It is widely used in digital cameras and cell phones for face detection and tracking.