Face Detection

Shree K. Nayar

Lecture: FPCV-2-5

Module: Features

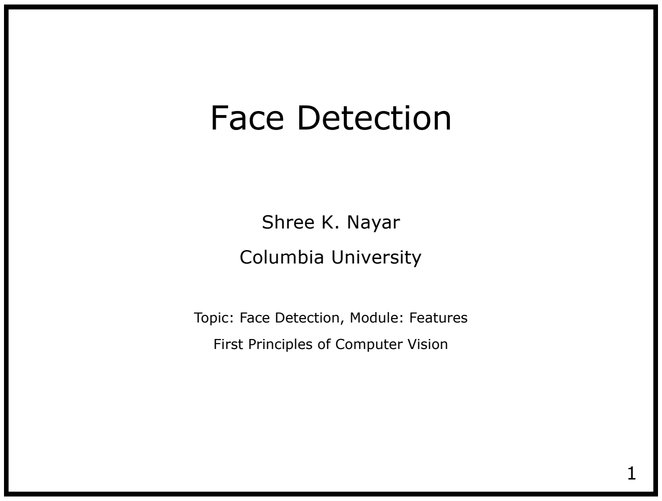
Series: First Principles of Computer Vision

Computer Science, Columbia University

February, 2025

FPCV Channel

FPCV Website



Face Detection

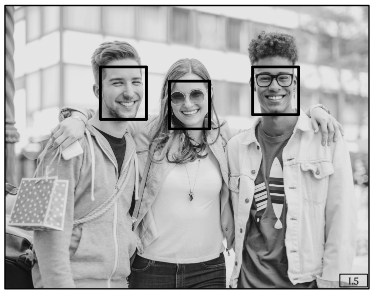
Shree K. Nayar
Columbia University

Topic: Face Detection, Module: Features
First Principles of Computer Vision

1

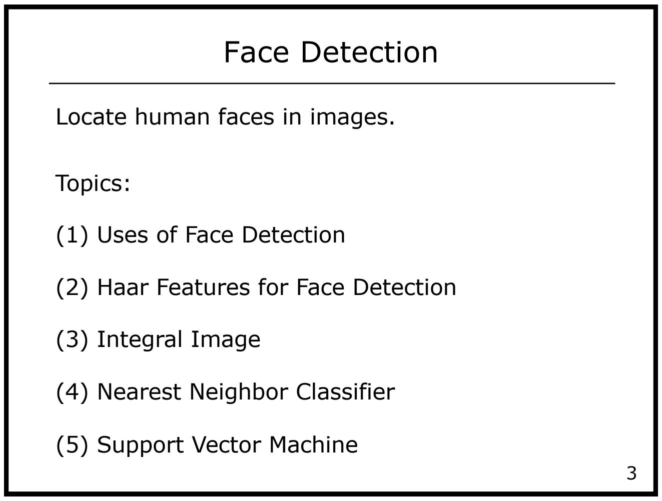
What is Face Detection?

Locate human faces in images



2

Today, face detection is a widely used computer vision technology. A typical face detector would place a window around each face in an image, as shown on the right. We want the detector to be able to find faces of different sizes because people could be at different distances from the camera. We would also like it to be able to handle rotations (pose) of the head with respect to the camera. Finally, it should be insensitive to the lighting of the face. In our discussion, we will keep things simple and assume that we're looking for frontal faces, where each face is viewed head-on by the camera, as in the image above.



Face Detection

Locate human faces in images.

Topics:

(1) Uses of Face Detection

(2) Haar Features for Face Detection

(3) Integral Image

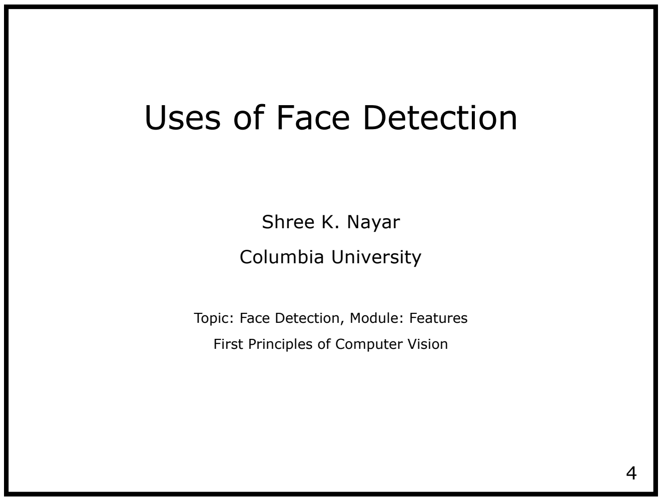
(4) Nearest Neighbor Classifier

(5) Support Vector Machine

3

We'll start by first looking at some important applications of face detection. Next, we will discuss what type of features would be good for face detection. We know how to compute edges, corners, and SIFT features. However, these features turn out not to be descriptive enough for reliably finding faces. Haar features, which are based on Haar filters, are particularly well-suited for face detection. In addition to being robust, they are also efficient to compute. The efficiency comes from the use of a representation called the integral image which is computed once for an image, and enables us to compute Haar features with a minimal number of computations. Interestingly, the computational cost of a Haar feature is independent of the size of the corresponding Harr filter.

We now have a feature vector at each pixel in the image, based on which we want to classify a window around the pixel as a face or a non-face. To solve the classification problem, we first look at nearest neighbor classifiers, where, given a vector, we want to find the nearest neighbor in feature space and simply assign the class of that neighbor to the pixel. While this approach is simple, it is computationally prohibitive as we need to repeat it for every pixel in the image, and there could be millions of pixels. A significantly more efficient approach is to pre-compute linear decision boundaries in feature space and use them to classify each feature vector based on which side of the boundary it falls on. The optimal decision boundaries are computed using an algorithm called the support vector machine. We show how a support vector machine can be constructed from a large set of feature vectors that correspond to faces and non-faces. We conclude with the results of applying our face detector to a video.



Uses of Face Detection

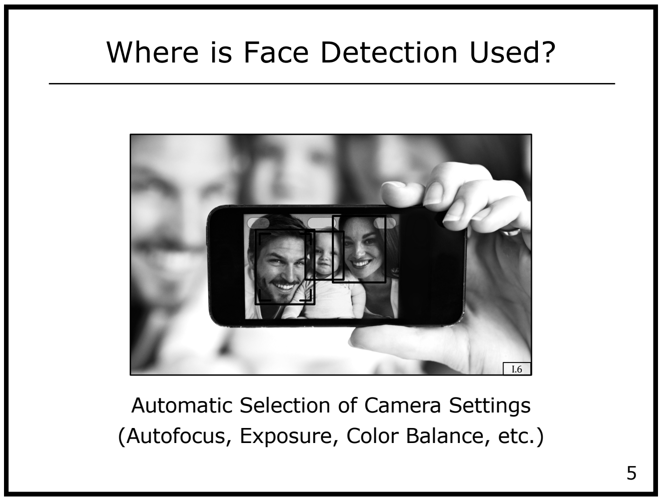
Shree K. Nayar

Columbia University

Topic: Face Detection, Module: Features

First Principles of Computer Vision

4



Where is Face Detection Used?

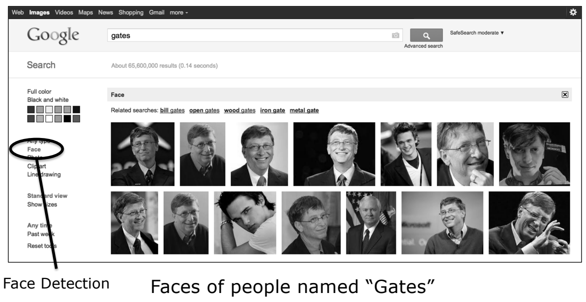
Automatic Selection of Camera Settings
(Autofocus, Exposure, Color Balance, etc.)

5

Let us look at some popular applications of face detection. One of them is the use of face detection in mobile phone cameras. When we turn on the camera app, we go into a preview mode in which face detection runs in real-time in the background. Since we typically want faces to be of high quality in photographs and videos, the parameters of the camera, such as the focus setting, exposure and color balancing parameters, are adjusted based on the quality of the faces in the image. Higher importance is typically given to the more prominent (larger) faces.

Here we see the application of visual search. In a search engine, if we run a query such as "gates", we will find physical gates and people with the name Gates. If we are only interested in people, then we can click the Face button (circled) which uses face detection to further filter the results. The end result is images of people with the name Gates.

Where is Face Detection Used?



Finding People using Search Engines

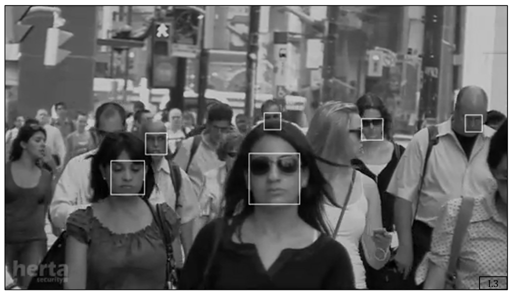
6

Where is Face Detection Used?

Intelligent Marketing

7

Where is Face Detection Used?

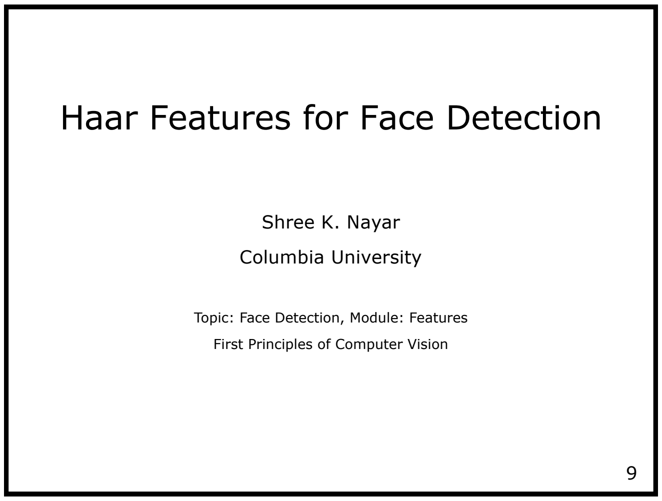


Biometrics, Surveillance, Monitoring

8

On the left is the application of face detection to intelligent marketing. Shown here is a vending machine on one of the platforms at Shinagawa Station in Japan. When you walk up to it, it estimates your gender and approximate age from your face, and based on this demographic information, it presents various products on the display that might be of interest to you.

Face detection is also used in malls to determine where people are spending time and attention. This information can be used to optimize inventory in a store or determine the price of advertising based on location. In the context of biometrics and security, shown on the right, face detection and recognition can be used to control access to spaces and to find suspicious people in public areas.

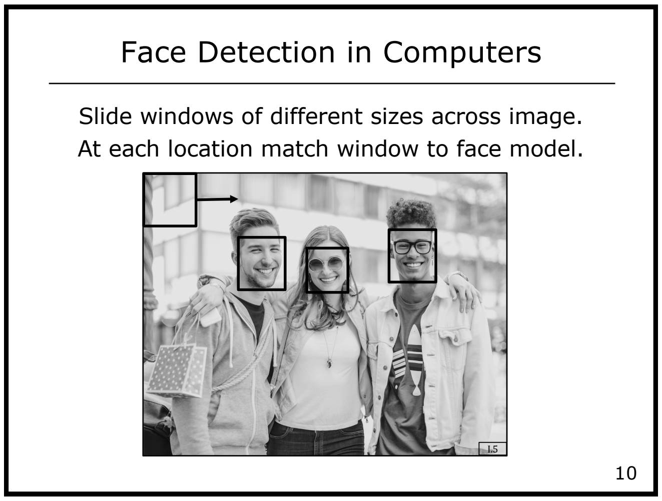


Haar Features for Face Detection

Shree K. Nayar
Columbia University

Topic: Face Detection, Module: Features
First Principles of Computer Vision

9



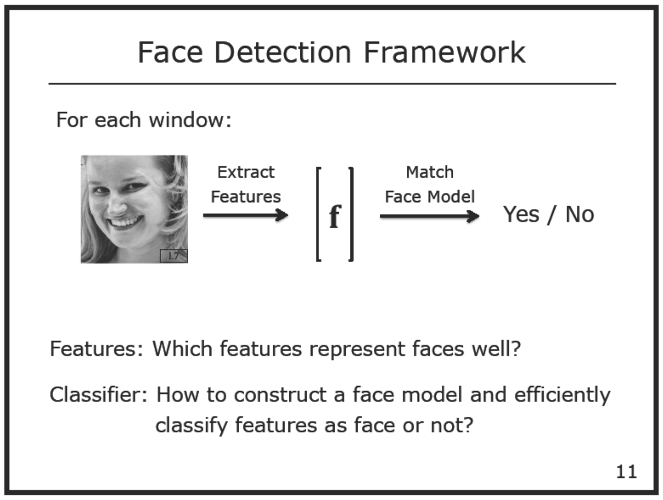
Face Detection in Computers

Slide windows of different sizes across image.
At each location match window to face model.

10

Now let's look at the problem of computing features for face detection. Imagine we wanted to find faces of a certain size. We would take a window of roughly that size and run it across the image in a raster scan fashion. At each pixel, we will extract features using the content within the window around it, and then classify the window as a face or a non-face. If we want to handle multiple scales or face sizes, we would repeat the above process for windows of different sizes.

Here is a window containing a face. We want to extract features from this window, to get a feature vector. Once we have this vector, we want use a classifier to classify the window as a face or a non-face.



Face Detection Framework

For each window:

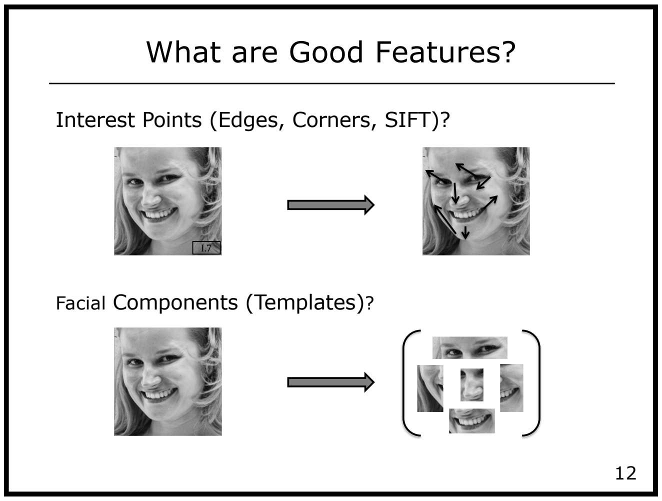
$$\xrightarrow{\text{Extract Features}} \begin{bmatrix} \mathbf{f} \end{bmatrix} \xrightarrow{\text{Match Face Model}} \text{Yes / No}$$

Features: Which features represent faces well?

Classifier: How to construct a face model and efficiently classify features as face or not?

11

What are good features for face detection? Edges and corners will not be that useful as they are not particularly descriptive. SIFT is an interesting candidate as it provides a description of the appearance within a window. While SIFT is very effective for matching appearances, in our context, the faces themselves can vary quite a bit in appearance, and what we are looking for is to distinguish between faces and non-faces. Another approach would be to use templates for different components of the face. That is, we could use templates for the eyes, the nose, the mouth, etc., and perform template matching within the window to detect each of these component. If all the components are detected, and they appear in a plausible configuration with respect to each other, then a face can be declared to exist in the window. Such an approach has been tried in the past, but with limited success.



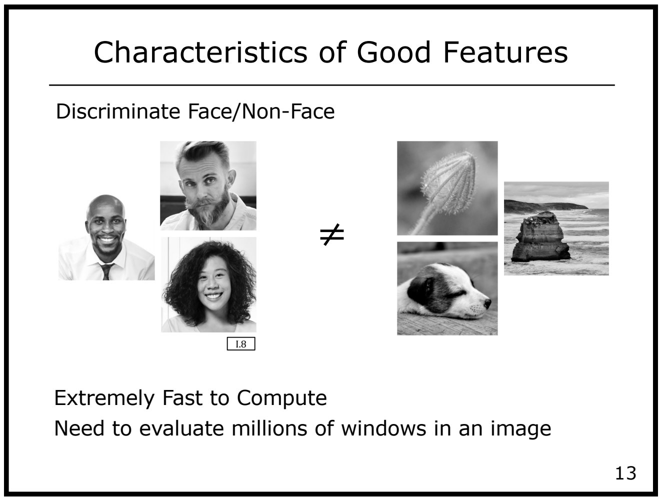
What are Good Features?

Interest Points (Edges, Corners, SIFT)?

Facial Components (Templates)?

12

As mentioned above, we need features that are good at discriminating between faces and non-faces. We also require the features to be extremely efficient to compute. An image could have millions of pixels and we need to perform feature extraction at each of these pixels. In one of the applications we discussed before, which is automatically adjusting camera parameters for photography, we need to have face detection running in the background in real-time. This requires face detection to be extremely fast.



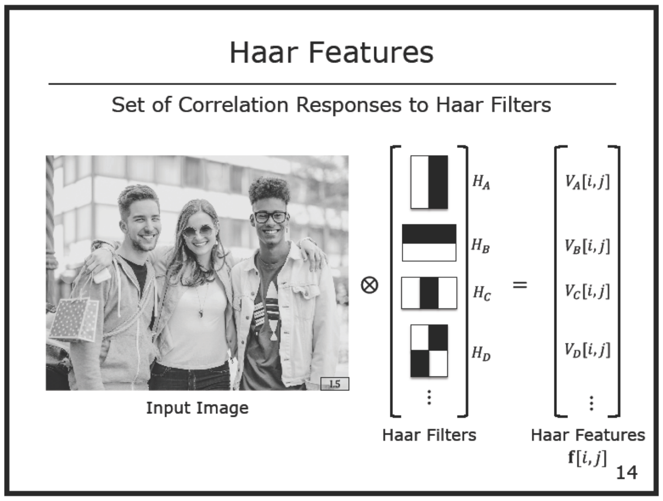
Characteristics of Good Features

Discriminate Face/Non-Face

Extremely Fast to Compute
Need to evaluate millions of windows in an image

13

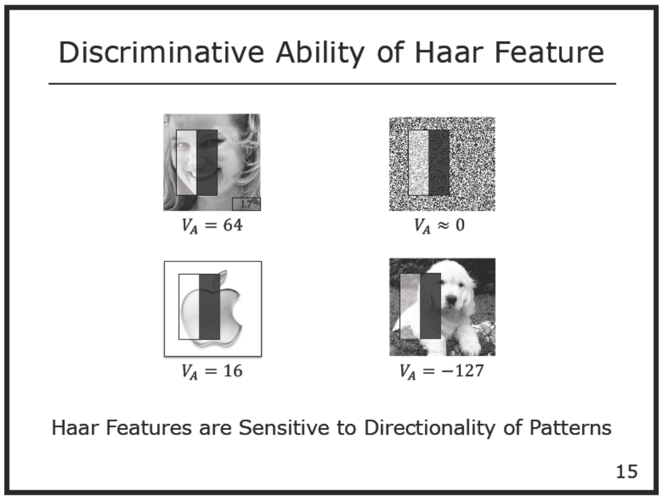
The features that we are going to use are called Haar features, which are computed using Haar filters. Haar filters are based on Haar wavelets, or essentially the square function. Shown here are the Haar filters for a specific window (face) size. Each filter is two-valued, which is computationally advantageous as we will see shortly. The entire set of filters yields a feature vector. Multiple such filter sets are used to find faces at different scales.



Haar Features

Set of Correlation Responses to Haar Filters

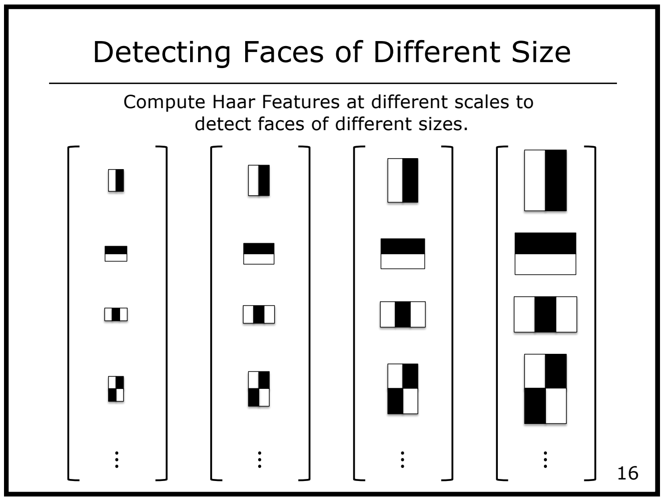
Shown here is the simplest Haar filter applied to the same pixel in different images. The output of the filter is shown below each image. Since the filter is white (value of 1) on one side and black (value of -1) on the other it acts much like a large first-order derivative filter.



Discriminative Ability of Haar Feature

Haar Features are Sensitive to Directionality of Patterns

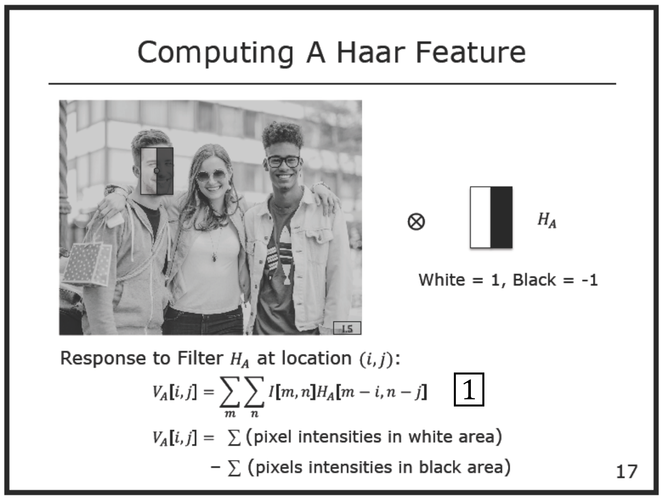
Here we have the complete set of Haar filters we use. Each column of filters corresponds to a particular scale. Within each column, the first two filters estimate derivatives in the X and Y directions, respectively. The third one resembles a Laplacian filter that finds the second derivative along the X direction. As we go further down the column, the filters represent higher order derivatives of the image. Our goal is to apply the entire set of filters shown here efficiently.



Detecting Faces of Different Size

Compute Haar Features at different scales to detect faces of different sizes.

Each Haar filter is applied to an image as a correlation operator. Let us take a closer look at the computations needed to apply the simplest of the Haar filters at a pixel. Remember that the white part of the filter corresponds to a value of 1, and the black part corresponds to a value of -1. As a result, we can find the result of the correlation $\boxed{1}$ at a pixel without any multiplications. Essentially, we subtract the sum of the pixel intensities in the black area from the sum of the pixel intensities in the white area. Additions and subtractions are significantly cheaper than multiplications and divisions, and this is one of the key advantages of using Haar filters.



Computing A Haar Feature

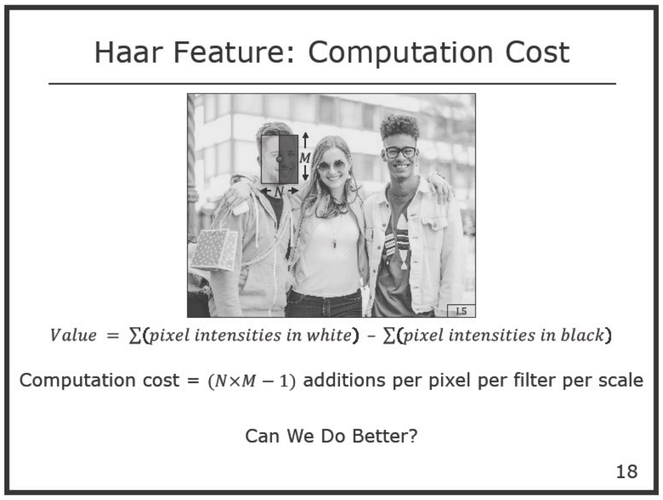
Response to Filter H_A at location (i, j):

$$V_A[i,j] = \sum_m \sum_n I[m,n] H_A[m-i, n-j] \quad \boxed{1}$$

$$V_A[i,j] = \sum \text{(pixel intensities in white area)} - \sum \text{(pixels intensities in black area)}$$

17

To determine the computational cost of applying the above filter, assume that the filter is of size *N* x *M*. Then, we need (*N* x *M* - 1) additions per pixel, per scale. While this is cheap, we have a large number of filters to apply at each pixel. Can we do even better in terms of computations?



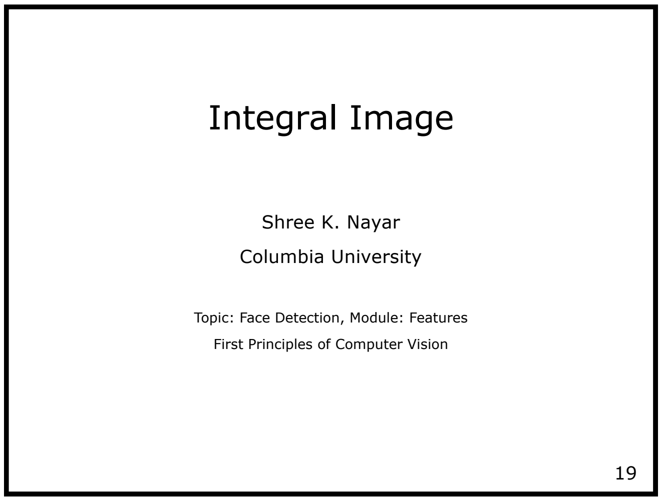
Haar Feature: Computation Cost

$$Value = \sum(pixel\ intensities\ in\ white) - \sum(pixel\ intensities\ in\ black)$$

Computation cost = $(N \times M - 1)$ additions per pixel per filter per scale

Can We Do Better?

18



Integral Image

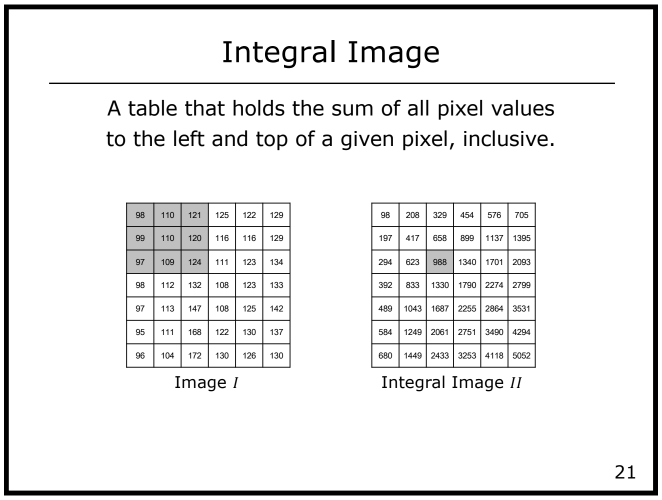
Shree K. Nayar

Columbia University

Topic: Face Detection, Module: Features

First Principles of Computer Vision

19



Integral Image

A table that holds the sum of all pixel values to the left and top of a given pixel, inclusive.

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image I

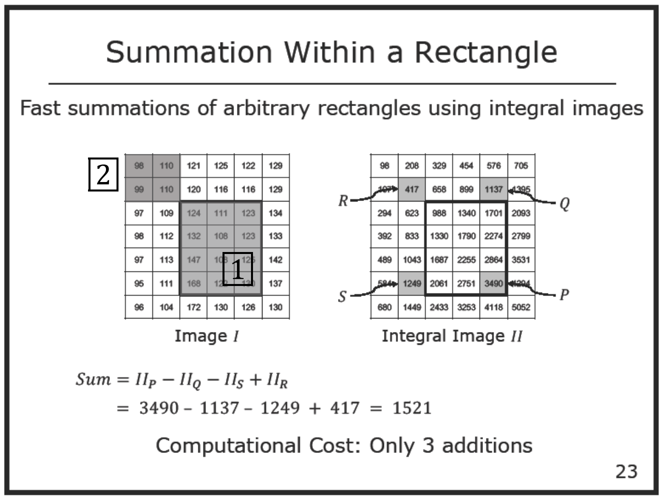
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

21

We use the concept of an integral image to make Haar features significantly more efficient to compute. On the right, we have an original image and its integral image. In the integral image, the value stored at a pixel is the sum of the values of the pixels in the original image that are above it and to the left of it, including itself. This is illustrated by the shaded pixel in the integral image and the shaded region in the original image used to compute its value.

Shown here is an image and its integral image. For now, let us assume the integral image is given to us; we will discuss how it is computed shortly. To find the sum of the values of pixels within the shaded box $\boxed{1}$ in the original image, we take the value at *P* in the integral image and subtract the values at *Q* and *S*. In doing so, we have subtracted the sum of pixels in the shaded area $\boxed{2}$ in the original image, twice. So, to the above, we add the integral image value at *R.* Hence, we have computed the sum of intensities within a rectangle in the original image with just three additions. It is worth noting that this cost is independent of the size of the rectangle.



Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images

Image I

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Integral Image II

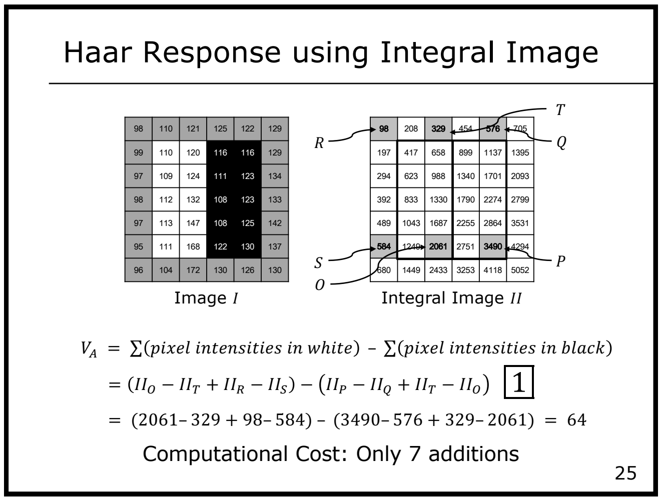
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

$$Sum = II_P - II_Q - II_S + II_R$$
$$= 3490 - 1137 - 1249 + 417 = 1521$$

Computational Cost: Only 3 additions

23

Now let's return to our Haar filters. Once again, we'll consider the simplest one, with half black and half white regions. Let's treat each half as a separate rectangle. Then, for the white region, we need the integral image values at *O*, *T*, *R*, and *S*. For the black region, we need the values at *P*, *Q*, *T*, and *O*. The result of applying the filter is the sum of intensities in the white region minus the sum of intensities in the black region $\boxed{1}$. Thus, we end up with seven additions to compute the filter output for each pixel. The more sophisticated filters in our bank will each have more black and white regions, which would necessitate more computations. Even so, the integral image makes each of the filters significantly more efficient to compute and this efficiency is independent of the size of the filter.



Haar Response using Integral Image

Image I

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Integral Image II

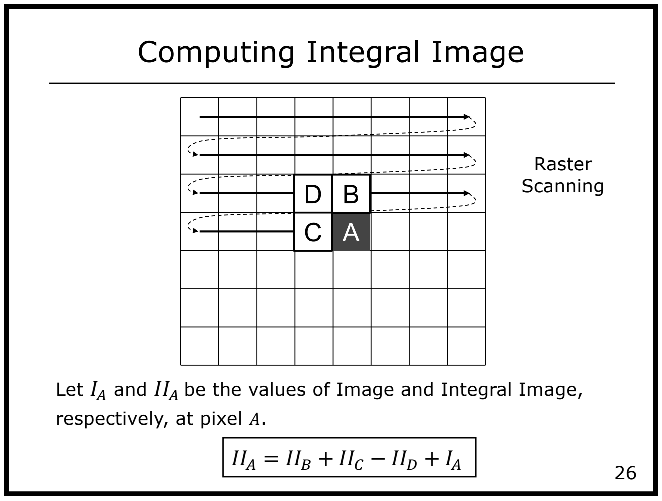
98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

$$V_A = \sum(pixel\ intensities\ in\ white) - \sum(pixel\ intensities\ in\ black)$$
$$= (II_O - II_T + II_R - II_S) - (II_P - II_Q + II_T - II_O) \quad \boxed{1}$$
$$= (2061 - 329 + 98 - 584) - (3490 - 576 + 329 - 2061) = 64$$

Computational Cost: Only 7 additions

25

Now, we describe how to compute the integral image of an image. This is done in a single raster scan of the image. The image is scanned starting from the top-left corner, and at each pixel A, we compute the value of the integral image using the integral image values at pixels B, C and D, all of which have already been computed. We simply add the integral image values at B and C and subtract the value at D to account for the fact that the rectangular region corresponding to D in the original image has been double counted.



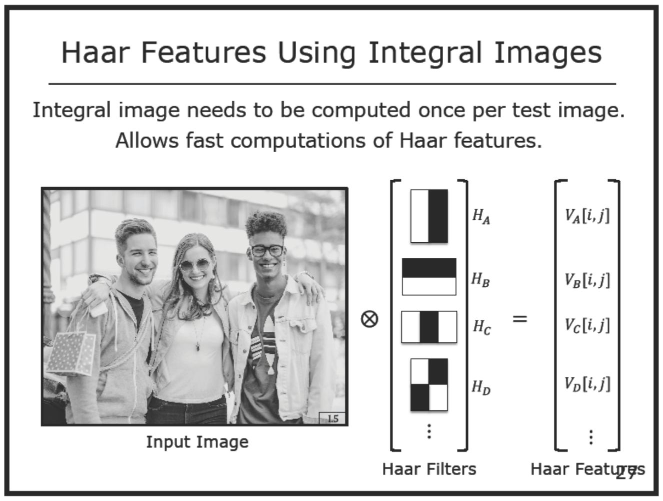
Computing Integral Image

Let I_A and II_A be the values of Image and Integral Image, respectively, at pixel A.

$$\boxed{II_A = II_B + II_C - II_D + I_A}$$

26

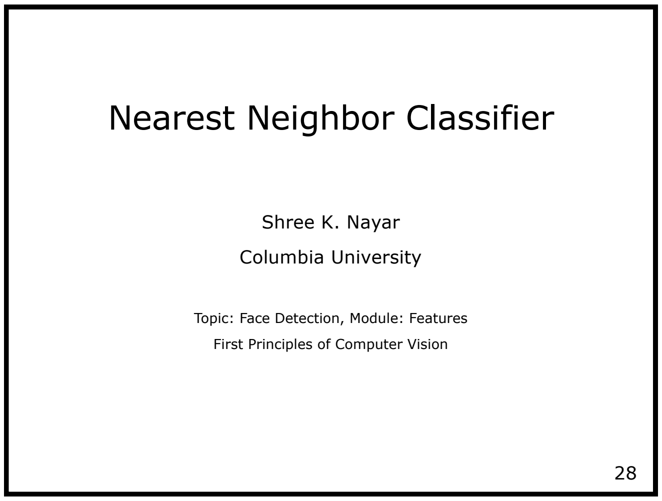
The integral image needs to be computed only once for a given image. Then, we can compute all the Haar features at each pixel to obtain the feature vector needed for classification. If we are interested in multiple scales, we will compute a feature vector and perform classification for each scale.



Haar Features Using Integral Images

Integral image needs to be computed once per test image. Allows fast computations of Haar features.

27



Nearest Neighbor Classifier

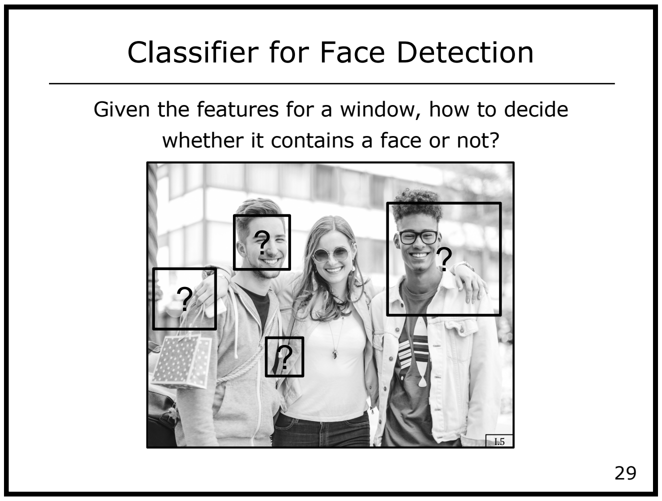
Shree K. Nayar

Columbia University

Topic: Face Detection, Module: Features

First Principles of Computer Vision

28

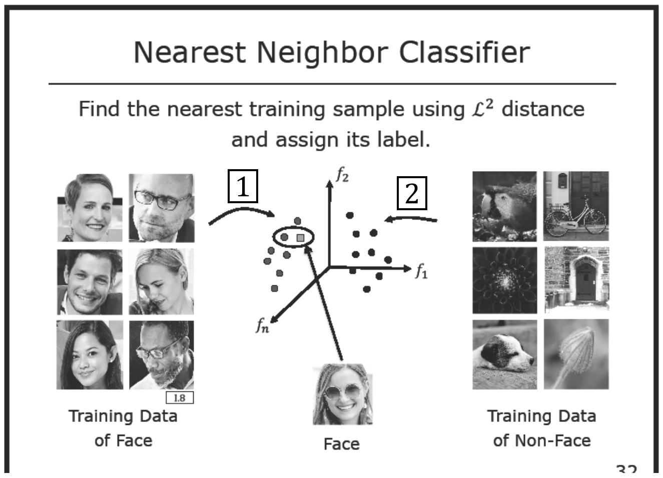


Classifier for Face Detection

Given the features for a window, how to decide whether it contains a face or not?

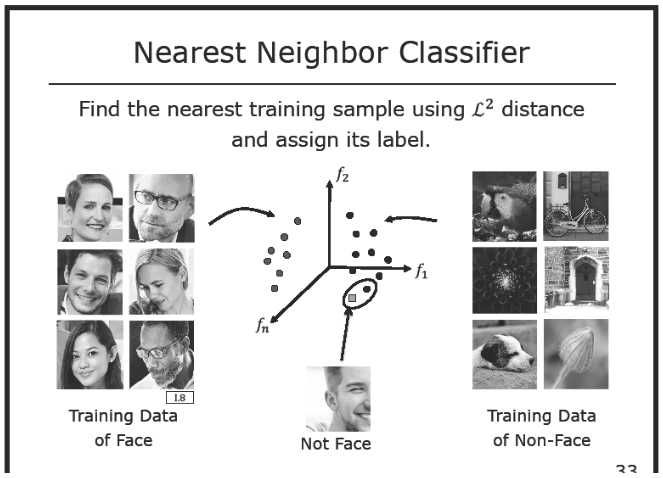
29

Now we are ready to perform classification. To construct our classifier, we will assume that we have many examples of faces and non-faces. This is what we refer to as training data in our context.



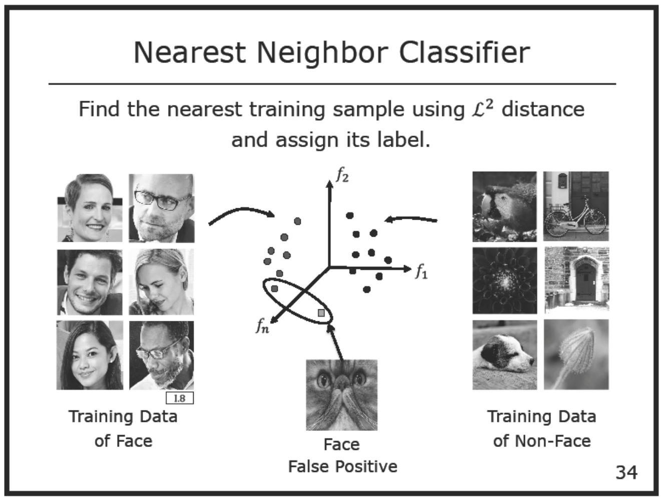
Nearest Neighbor Classifier

Find the nearest training sample using $\mathcal{L}^2$ distance and assign its label.



Nearest Neighbor Classifier

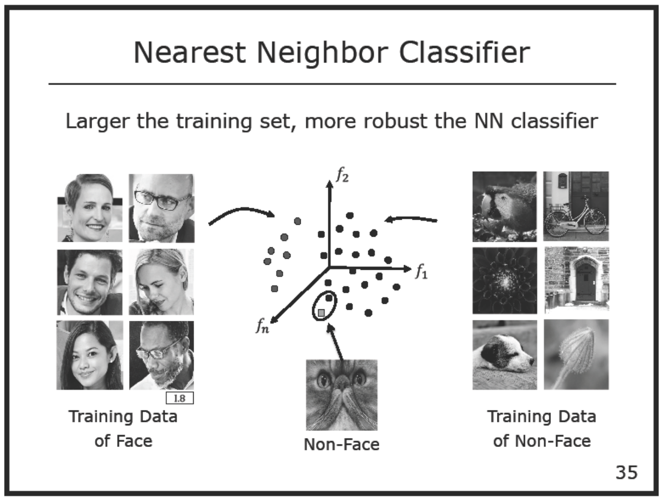
Find the nearest training sample using $\mathcal{L}^2$ distance and assign its label.



Nearest Neighbor Classifier

Find the nearest training sample using $\mathcal{L}^2$ distance and assign its label.

34



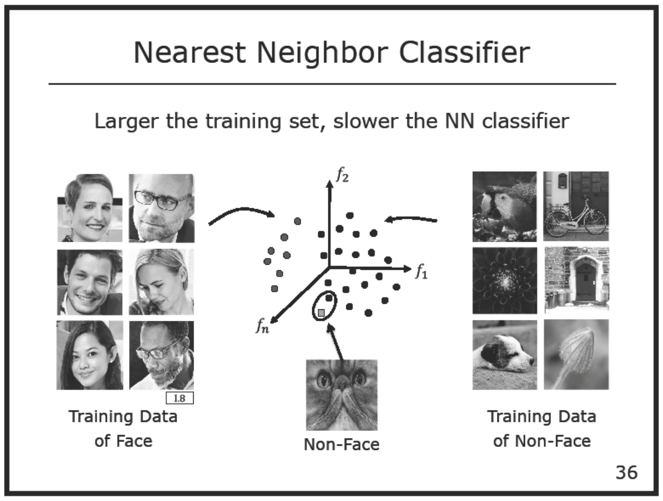
Nearest Neighbor Classifier

Larger the training set, more robust the NN classifier

35

Consider a Haar feature vector which has n elements. We can represent the feature as a point in n-dimensional space. In slide 32, the training points on the left correspond to faces $\boxed{1}$,and the ones on the right correspond to non-faces $\boxed{2}$. Given a new image window, we compute the feature vector for the window, which is again a point in n-dimensional space. We can find the closest neighbor in our training data to the new feature and assign it the class of this neighbor. This is how a nearest neighbor classifier works. In slide 32, we see that the novel window is that of a face and it is correctly classified as such. In slide 33, the input is a part of a face, which is not a face, and hence classified as a non-face. In the case of the cat in slide 34, the image is face-like and hence its nearest neighbor turns out to be a face. We clearly do not want to classify the cat as a human face. This is referred to as a false positive. Note that the nearest neighbor in this case is not all that close to the input feature vector. One way to avoid such errors in classification is to increase our training data. If we substantially increase the non-faces, as in slide 35, it becomes more likely that such face-like (but not face) images will be classified as non-faces.

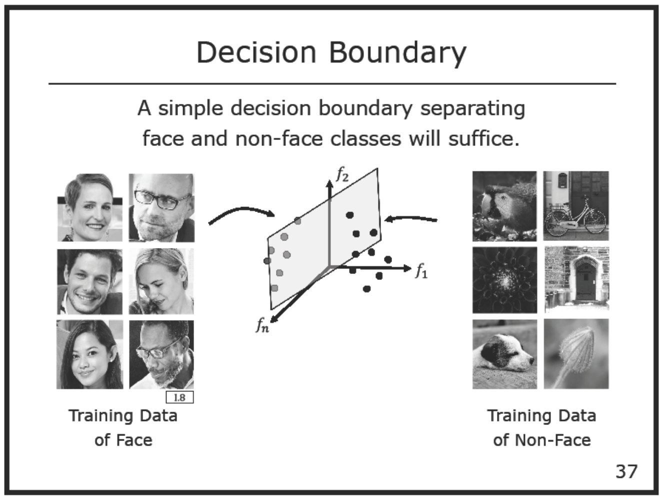
We have seen that for the detector to perform well, we will need a large set of face and non-face examples. The problem with using a nearest neighbor classifier is the cost of finding the nearest neighbor. The naïve approach is of course to compare the input features with each of the training features. There are smarter ways of performing the search for the nearest neighbor, such as the use of K-D trees. However, even these approaches are expensive given the large number of pixels and features we are dealing with.



Nearest Neighbor Classifier

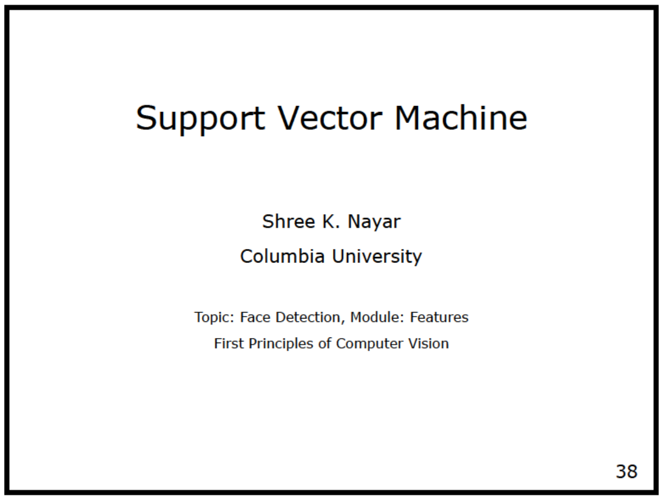
Larger the training set, slower the NN classifier

That brings us to decision boundaries. The above problem becomes much simpler if, somehow, we can place a boundary between the two clusters—faces and non-faces. Then, all we need to do to classify a new window is to check which side of the boundary its feature lies on.



Decision Boundary

A simple decision boundary separating face and non-face classes will suffice.



Support Vector Machine

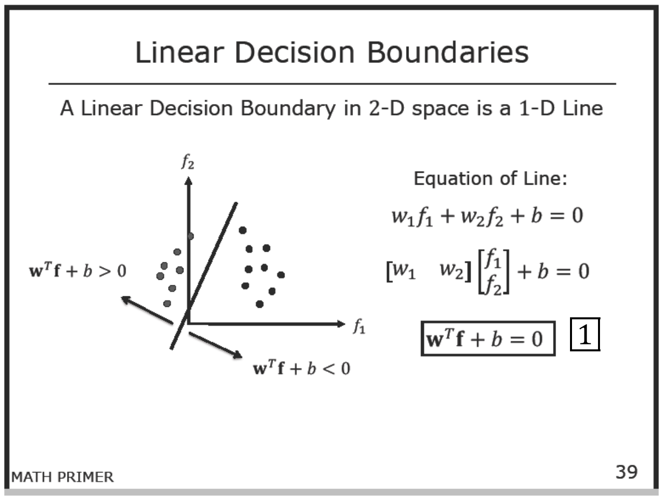
Shree K. Nayar

Columbia University

Topic: Face Detection, Module: Features

First Principles of Computer Vision

38



Linear Decision Boundaries

A Linear Decision Boundary in 2-D space is a 1-D Line

Equation of Line:

$$w_1 f_1 + w_2 f_2 + b = 0$$

$$[w_1 \quad w_2]\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + b = 0$$

$$\boxed{\mathbf{w}^T\mathbf{f}+b=0} \quad \boxed{1}$$

MATH PRIMER 39

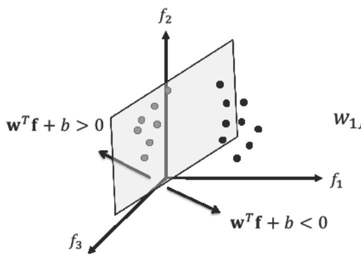
We now present the concept of a support vector machine (SVM), which uses optimal decision boundaries to perform classification. SVMs are widely used not just in computer vision but in many different fields.

Let us assume that our feature vector is two-dimensional. The decision boundary in this case is a line. In vector form, the equation of a line is the coefficient vector **w** times the feature vector **f** plus a scalar b $\boxed{1}$. Now, given a new feature **f**, we simply plug it into the left side of the equation, and if the left side it is greater than 0, it is a face, and if it is less than 0, it is not a face.

In a three-dimensional feature space, the linear decision boundary would be a 2D plane. Once again, we can write the equation of the plane using the same form as before $\boxed{1}$. The parameters **w** and b describe the plane. For a new feature **f,** the classification test remains the same as before.

Linear Decision Boundaries

A Linear Decision Boundary in 3-D space is a 2-D Plane


f_2
f_1
f_3
$\mathbf{w}^T\mathbf{f} + b > 0$
$\mathbf{w}^T\mathbf{f} + b < 0$

Equation of Plane:

$$w_1 f_1 + w_2 f_2 + w_3 f_3 + b = 0$$

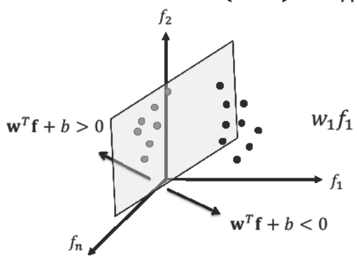
$$\mathbf{w}^T\mathbf{f} + b = 0 \quad \boxed{1}$$

MATH PRIMER 40

Linear decision boundaries of the same form can be used irrespective of the dimensionality of the feature space. In an n-dimensional feature space, we have an (n-1)-dimensional hyperplane. Once again, the same equation and test can be used for classification.

Linear Decision Boundaries

A Linear Decision Boundary in n-D space is a $(n-1)$-D Hyperplane


f_2
f_1
f_n
$\mathbf{w}^T\mathbf{f} + b > 0$
$\mathbf{w}^T\mathbf{f} + b < 0$

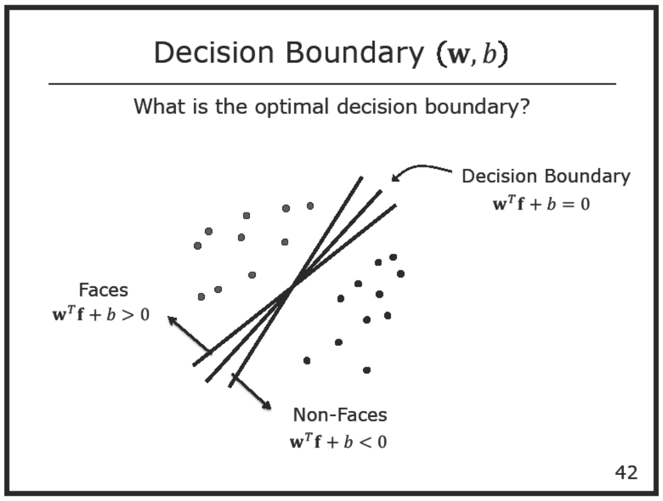
Equation of Hyperplane:

$$w_1 f_1 + w_2 f_2 + \cdots + w_n f_n + b = 0$$

$$\mathbf{w}^T\mathbf{f} + b = 0$$

MATH PRIMER 41

The key question is then, what is the optimal decision boundary to use, given a collection of faces and non-faces? In the example shown here, all of the three boundaries partition the face and non-face data well. However, when given a new input feature, the classification result can depend on which of these boundaries we choose.

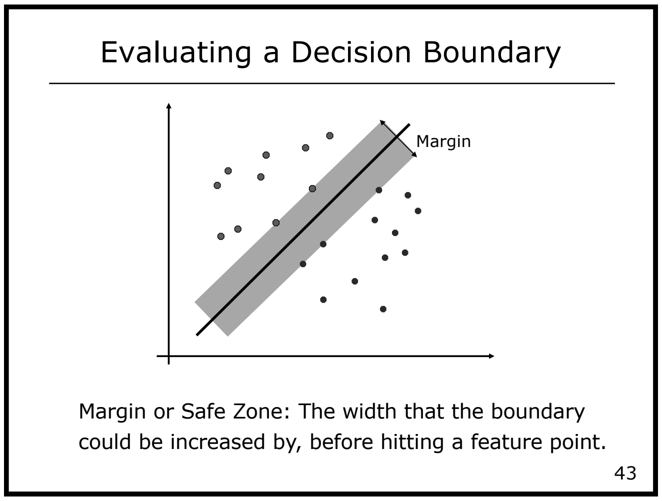

Decision Boundary $(\mathbf{w}, b)$

What is the optimal decision boundary?

Decision Boundary
$\mathbf{w}^T\mathbf{f} + b = 0$
Faces
$\mathbf{w}^T\mathbf{f} + b > 0$
Non-Faces
$\mathbf{w}^T\mathbf{f} + b < 0$

42

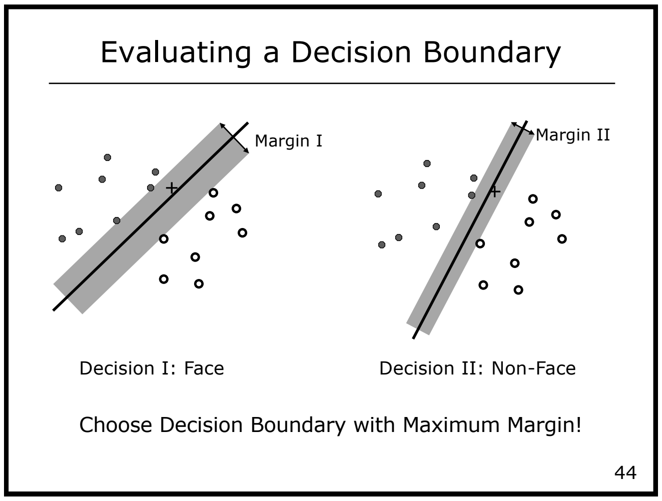
To evaluate the quality of a decision boundary, we define what is called a safe zone (shaded region). The width of the safe zone is called the margin. It is the width to which we can thicken the decision boundary until it touches features on both sides. Here, we've extended the safe zone until it hits two points on one side and the three points on the other. In general, the margin is therefore the width that a boundary can be increased by before it hits at least one feature point on each of the two sides.


Evaluating a Decision Boundary

Margin or Safe Zone: The width that the boundary could be increased by, before hitting a feature point.

43

We can see that different decision boundaries will produce different safe zones. We want to choose the one that maximizes the margin, and that's achieved by using the support vector machine.


Evaluating a Decision Boundary

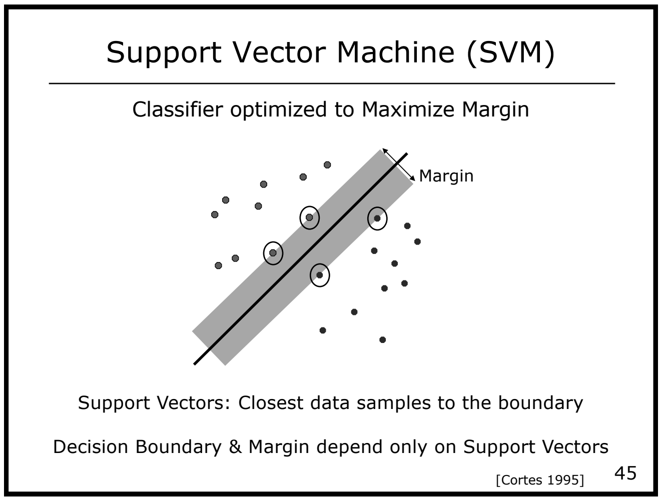
Decision I: Face

Decision II: Non-Face

Choose Decision Boundary with Maximum Margin!

44

It is called a support vector machine (SVM) because, when we are at the maximum margin, we are touching the points that can be seen as supporting the safe zone. Interestingly, once we have computed the support vectors and know which points lie on the boundary of the safe zone, we can safely ignore all the other points in our training set.



Support Vector Machine (SVM)

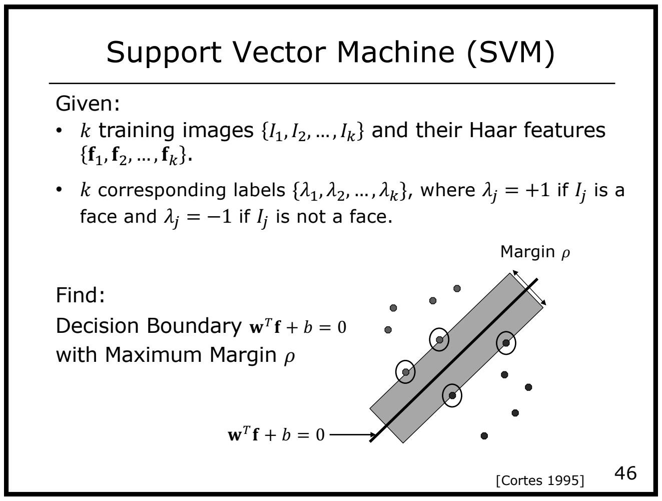
Classifier optimized to Maximize Margin

Support Vectors: Closest data samples to the boundary

Decision Boundary & Margin depend only on Support Vectors

[Cortes 1995] 45

Let us now look at how we can compute the support vectors. We describe the method for a single scale, noting that each scale requires a separate classifier. Given a set of training images $\{I_1, I_2, \dots I_k\}$ we compute their Haar features $\{\mathbf{f_1}, \mathbf{f_2}, \dots \mathbf{f_k}\}$. Since these are training images, we know their labels $\{\lambda_1, \lambda_2, \dots \lambda_k\}$, where a label is +1 for face and -1 for non-face. We want to find the decision boundary that maximizes the margin ρ.



Support Vector Machine (SVM)

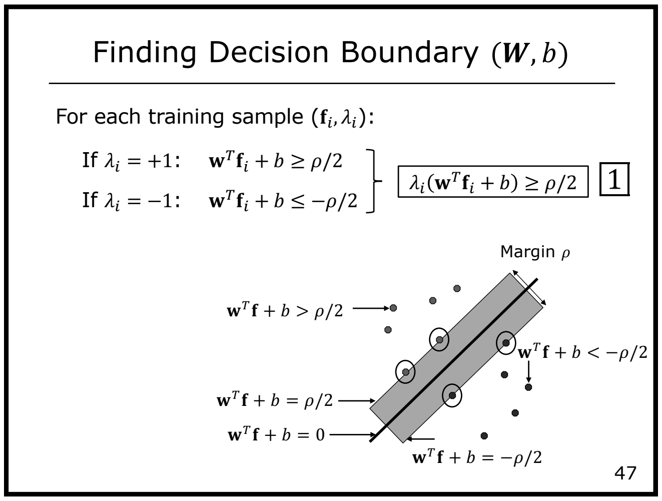
Given:

- k training images $\{I_1, I_2, \dots, I_k\}$ and their Haar features $\{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_k\}$.
- k corresponding labels $\{\lambda_1, \lambda_2, \dots, \lambda_k\}$, where $\lambda_j = +1$ if I_j is a face and $\lambda_j = -1$ if I_j is not a face.

Find:
Decision Boundary $\mathbf{w}^T\mathbf{f} + b = 0$
with Maximum Margin ρ

[Cortes 1995] 46

For each training sample, if the label is +1, then it lies to the left of the margin, and if it is -1, it lies to the right. We can combine both these constraints into one expression $\boxed{1}$.



Finding Decision Boundary $(\boldsymbol{W}, b)$

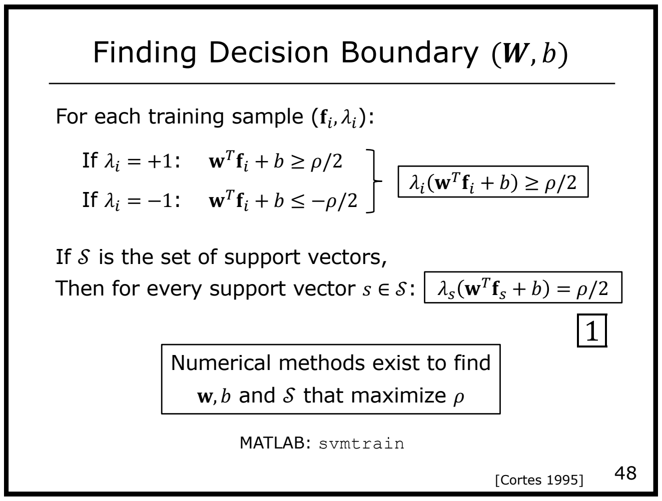
For each training sample $(\mathbf{f}_i, \lambda_i)$:

$$\left.\begin{array}{ll} \text{If } \lambda_i = +1: & \mathbf{w}^T\mathbf{f}_i + b \geq \rho/2 \\ \text{If } \lambda_i = -1: & \mathbf{w}^T\mathbf{f}_i + b \leq -\rho/2 \end{array}\right\} \quad \boxed{\lambda_i(\mathbf{w}^T\mathbf{f}_i + b) \geq \rho/2} \quad \boxed{1}$$

Margin ρ
$\mathbf{w}^T\mathbf{f} + b > \rho/2$
$\mathbf{w}^T\mathbf{f} + b < -\rho/2$
$\mathbf{w}^T\mathbf{f} + b = \rho/2$
$\mathbf{w}^T\mathbf{f} + b = 0$
$\mathbf{w}^T\mathbf{f} + b = -\rho/2$

47

Therefore, for a support vector (a vector touching the safe zone), we get equation $\boxed{1}$. In other words, support vectors must be exactly a distance of $\rho/2$ from the decision boundary. There are well known algorithms for finding the parameters **w** and b from training data. An example is the function `svmtrain` in MATLAB.



Finding Decision Boundary $(\boldsymbol{W}, b)$

For each training sample $(\mathbf{f}_i, \lambda_i)$:

$$\left.\begin{array}{ll} \text{If } \lambda_i = +1: & \mathbf{w}^T\mathbf{f}_i + b \geq \rho/2 \\ \text{If } \lambda_i = -1: & \mathbf{w}^T\mathbf{f}_i + b \leq -\rho/2 \end{array}\right\} \boxed{\lambda_i(\mathbf{w}^T\mathbf{f}_i + b) \geq \rho/2}$$

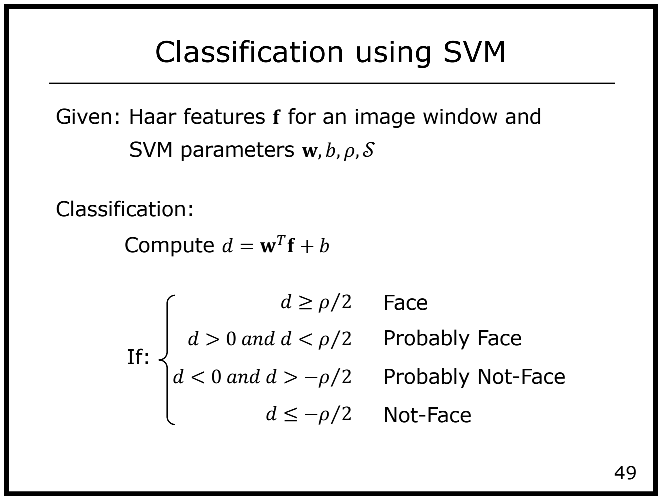
If $\mathcal{S}$ is the set of support vectors,
Then for every support vector $s \in \mathcal{S}$: $\boxed{\lambda_s(\mathbf{w}^T\mathbf{f}_s + b) = \rho/2}$ $\boxed{1}$

Numerical methods exist to find $\mathbf{w}, b$ and $\mathcal{S}$ that maximize ρ

MATLAB: `svmtrain`

[Cortes 1995] 48

Now we can classify a new feature vector **f** based on the decision boundary we just computed. We simply compute the distance *d* of the feature from the decision boundary. If *d* is greater or equal to $\rho/2$, it is a face; and if it is less or equal to $-\rho/2$, it is not a face. If **f** lies within the safe zone, we can classify it as being probably a face or probably not a face.



Classification using SVM

Given: Haar features $\mathbf{f}$ for an image window and SVM parameters $\mathbf{w}, b, \rho, \mathcal{S}$

Classification:

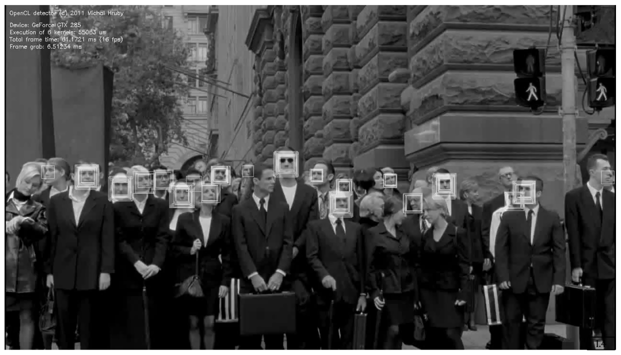
Compute $d = \mathbf{w}^T\mathbf{f} + b$

$$\text{If: } \begin{cases} d \geq \rho/2 & \text{Face} \\ d > 0 \; and \; d < \rho/2 & \text{Probably Face} \\ d < 0 \; and \; d > -\rho/2 & \text{Probably Not-Face} \\ d \leq -\rho/2 & \text{Not-Face} \end{cases}$$

49

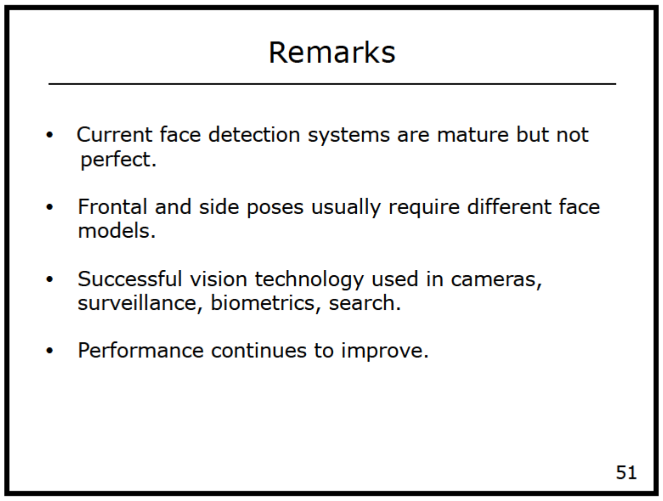
In this example provided by Mikhail Hruby, the above face detector is applied to a clip from the movie *The Matrix*. We end up with multiple windows around each face because there are several pixels or scales that are close to each other that produce features that are classified as faces. One can imagine using something like non-maximum suppression, which we discussed in the context of corner detection, to find a single window to represent each cluster of windows.

Face Detection Results



50

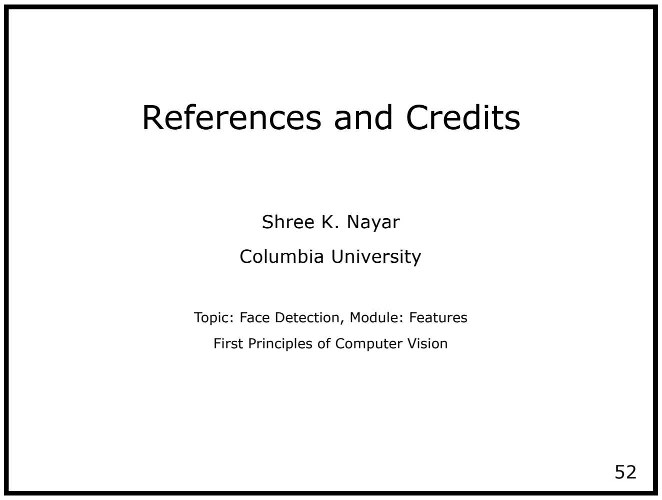
We will conclude with a few remarks related to face detection. First, while current face detection systems are not perfect, they work very well. We know this as they are being widely used today in a variety of products. Second, one way to handle non-frontal faces (faces oriented away from the camera) is to train additional classifiers, each one tuned to handle a small range of orientations. Face detection is a highly successful vision technology that is used in photographic cameras, surveillance, security, biometrics, and so on. Finally, the performance of face detection continues to improve. In fact, if we consider face recognition, for which face detection is a precursor, machines are beginning to surpass the performance of humans.



Remarks

- Current face detection systems are mature but not perfect.
- Frontal and side poses usually require different face models.
- Successful vision technology used in cameras, surveillance, biometrics, search.
- Performance continues to improve.

51



References and Credits

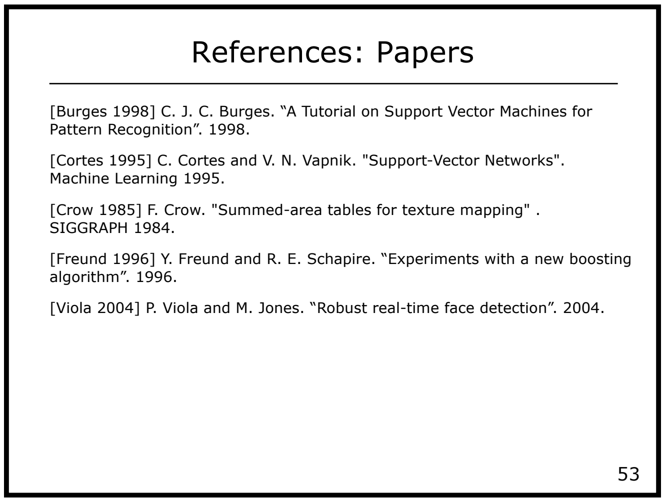
Shree K. Nayar

Columbia University

Topic: Face Detection, Module: Features

First Principles of Computer Vision

52



References: Papers

[Burges 1998] C. J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". 1998.

[Cortes 1995] C. Cortes and V. N. Vapnik. "Support-Vector Networks". Machine Learning 1995.

[Crow 1985] F. Crow. "Summed-area tables for texture mapping" . SIGGRAPH 1984.

[Freund 1996] Y. Freund and R. E. Schapire. "Experiments with a new boosting algorithm". 1996.

[Viola 2004] P. Viola and M. Jones. "Robust real-time face detection". 2004.

53

Image Credits

I.1 http://dailymail.co.uk/sciencetech/article-1339112/Facebook-facial-recognition-. software-suggest-friends-tagging-new-photos.html Associated Newspapers Ltd.
I.2 http://www.designboom.com/design/acure-digital-vending-machine/
I.3 https://www.youtube.com/watch?v=meRSKCS0d-A Herta Security.
I.4 M. Hruby. http://mhr3.blogspot.com/2012/03/face-detection-with-opencl.html Used with permission.
I.5 Purchased from iStock by Getty Images.
I.6 Purchased from iStock by Getty Images.
I.7 Purchased from iStock by Getty Images.
I.8 Purchased from iStock by Getty Images.

54

Acknowledgements: Thanks to Roshan Kenia and Ayush Sharma for their help with transcription, editing and proofreading.

References

[Burges 1998] C. J. C. Burges. “A Tutorial on Support Vector Machines for Pattern Recognition”. 1998.

[Cortes 1995] C. Cortes and V. N. Vapnik. "Support-Vector Networks". Machine Learning 1995.

[Crow 1985] F. Crow. "Summed-area tables for texture mapping" . SIGGRAPH 1984.

[Freund 1996] Y. Freund and R. E. Schapire. “Experiments with a new boosting algorithm”. 1996.

[Viola 2004] P. Viola and M. Jones. “Robust real-time face detection”. 2004.