



# OBJECT DETECTION

## 101



# What is Object Detection?

# OBJECT DETECTION

Object Detection is a Computer Vision technique used to identify instances (occurrence) of objects in images or videos.





# Object Detection Objectives:

- Determine whether a particular object is present in the image.
- Locate the object in the image.
- Identify the class of the object.

## Use Cases:

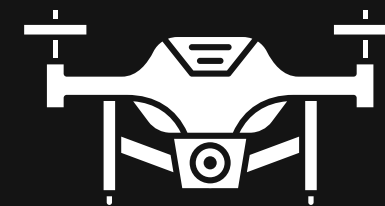
Self - Driving Cars



Medical Diagnosis



Face Recognition



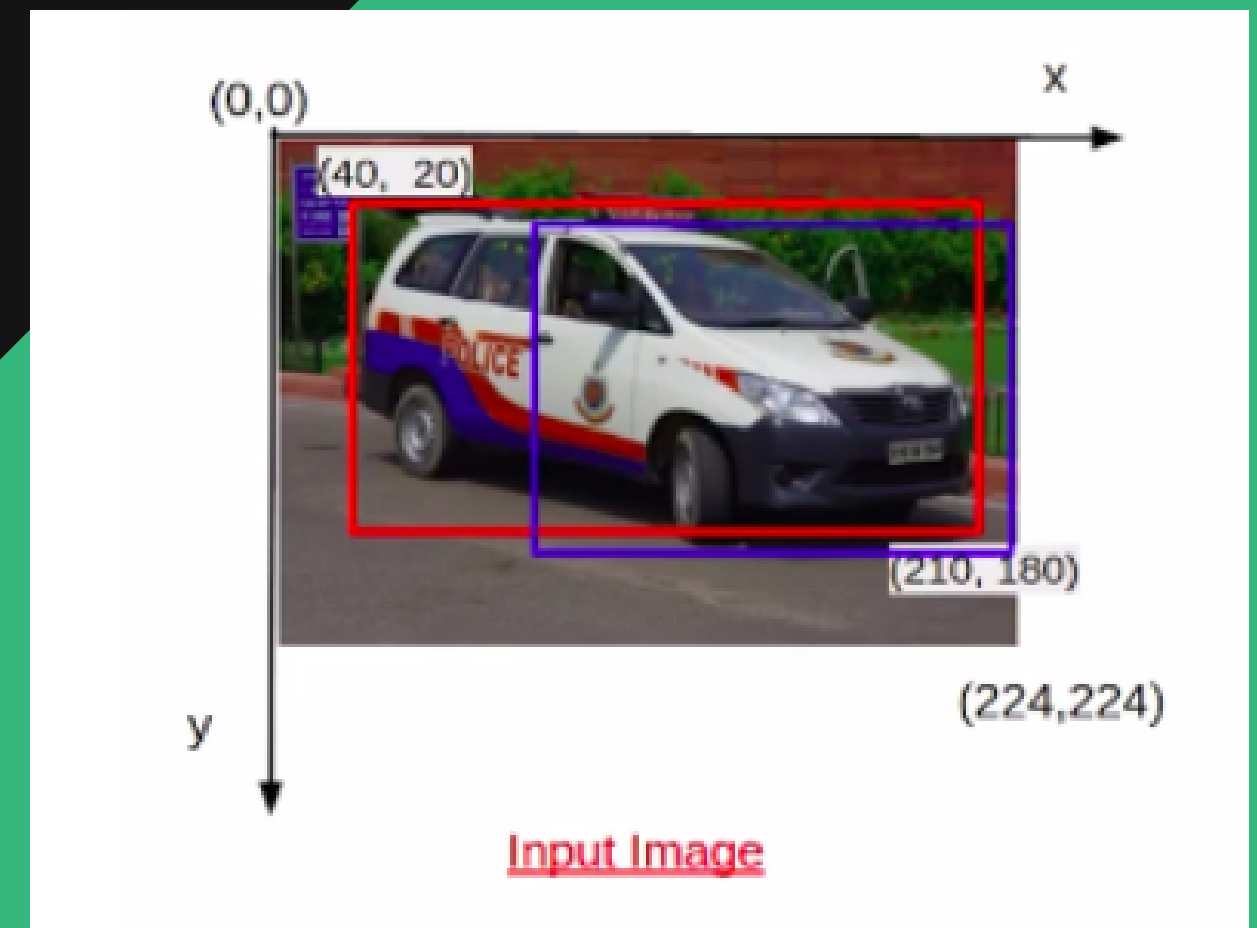
Surveillance

# HOW IS THE DATA ACTUALLY STORED??

Data is stored in a numerical form and holds the following values:

1. The probability of the object being present in the image
2. The x and y coordinates of the bounding box
3. The class value of the object.

(only applicable for multi class object detection cases.)



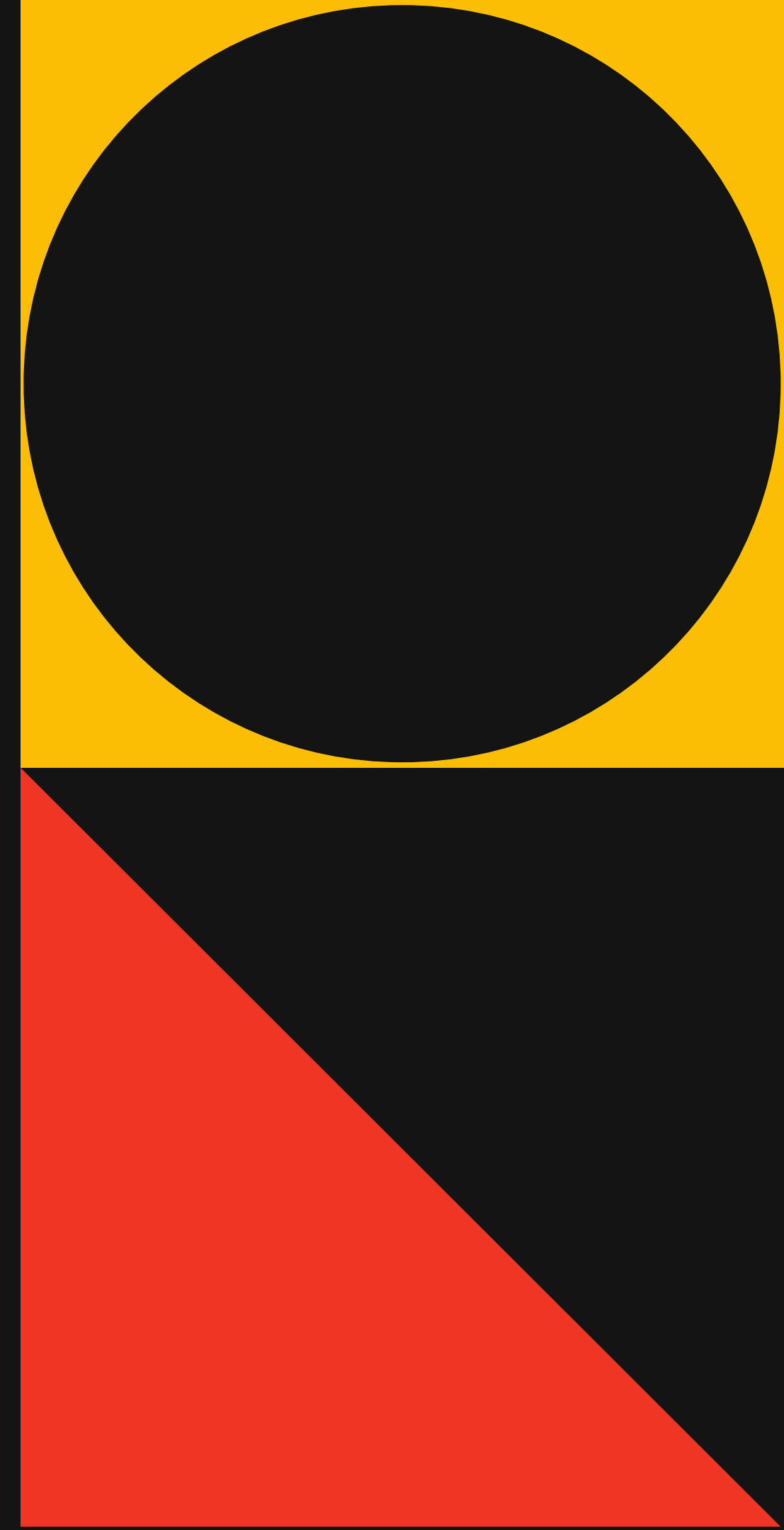


Let's dive deep into the  
important techniques  
useful in Object Detection

# Haar Cascades

- Haar Cascades is an algorithm that can detect objects in images, irrespective of their scale and location in the image.
- It is extremely useful in face detection.
- Haar Cascades are fast, easy to implement and thus work well in real-time with less computing power.

The classifier model based on the Haar Cascades algorithm is trained on a lot of positive and negative input images and learns to classify images containing a face and not containing a face. How does it do that?



# Haar Features

A Haar Feature is a rectangular black-and-white patch or kernel.

The black region corresponds to pixels of intensity 1.0 and white region corresponds to pixels of intensities 0.0

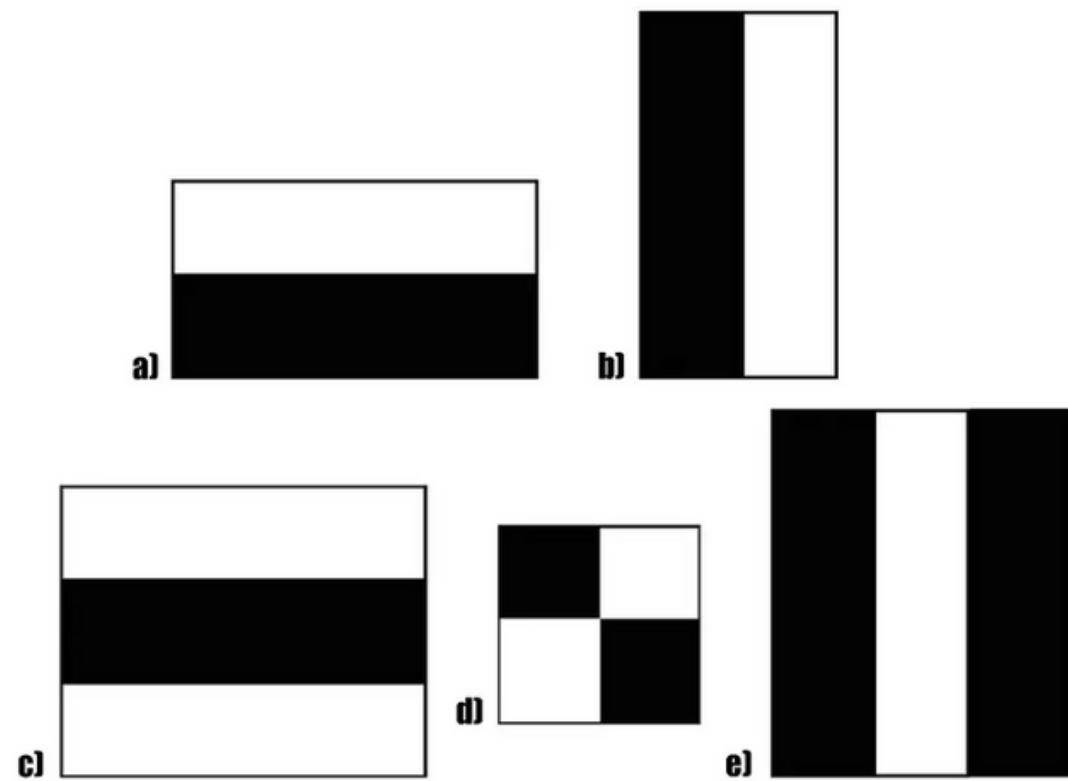


Fig. A sample of Haar features used in the Original Research Paper published by Viola and Jones.

The regions in the Haar feature can be of varying sizes. Different Haar Features are used at various stages of the algorithm.



# Application of Haar Features

0.4	0.7	0.9	0.7	0.4	0.5	1.0	0.3
0.3	1.0	0.5	0.8	0.7	0.4	0.1	0.4
0.9	0.4	0.1	0.2	0.5	0.8	0.2	0.9
0.3	0.5	0.8	1.0	0.3	0.7	0.5	0.3
0.2	0.9	0.1	0.5	0.1	0.4	0.8	0.8
0.5	0.1	0.3	0.7	0.9	0.6	1.0	0.2
0.8	0.4	1.0	0.2	0.7	0.3	0.1	0.4
0.4	0.9	0.6	0.6	0.2	1.0	0.5	0.9

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

=

SUM OF THE DARK PIXELS/NUMBER OF DARK PIXELS -  
SUM OF THE LIGHT PIXELS/NUMBER OF THE LIGHT PIXELS

$(0.7 + 0.4 + 0.1 + 0.5 + 0.8 + 0.2 + 0.3 + 0.7 + 0.5 +$   
 $0.1 + 0.4 + 0.8 + 0.9 + 0.6 + 1.0 + 0.7 + 0.3 + 0.1)/18$

$(1.0 + 0.5 + 0.8 + 0.4 + 0.1 + 0.2 + 0.6 + 0.8 + 1.0 +$   
 $0.9 + 0.1 + 0.5 + 0.1 + 0.3 + 0.7 + 0.4 + 1.0 + 0.2)/18$

$0.51 - 0.53 = -0.02$

The pixel intensities in each region are added and then, the sum corresponding to the dark and light pixels are subtracted to get the Haar value.

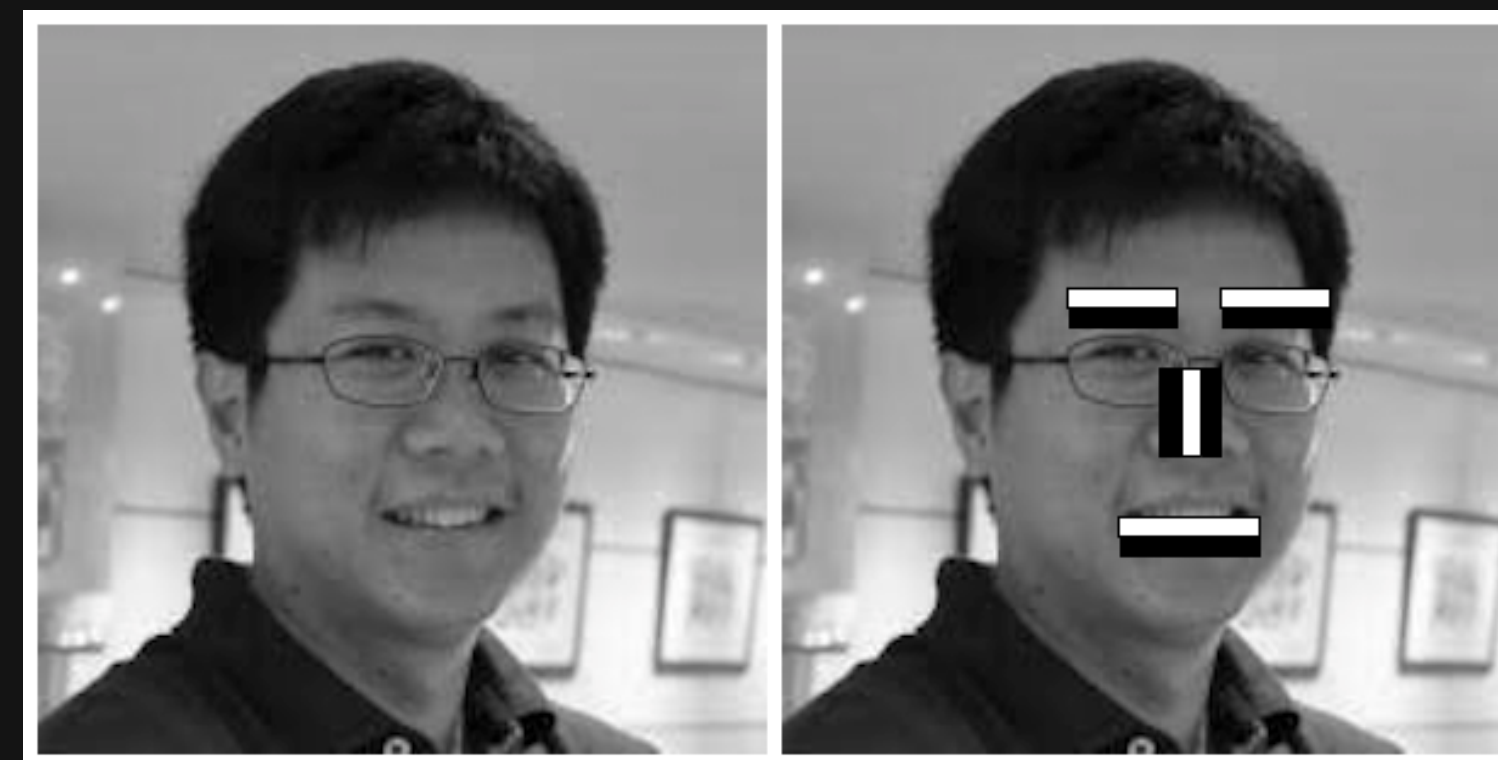


**But how do we use these to detect faces??**



- This is how Haar Features traverse over the entire image.
- Usually, a subset of the image is considered over which different Haar features of varying pixel sizes are passed to detect certain features of the face, like eyebrows, nose, lips etc.

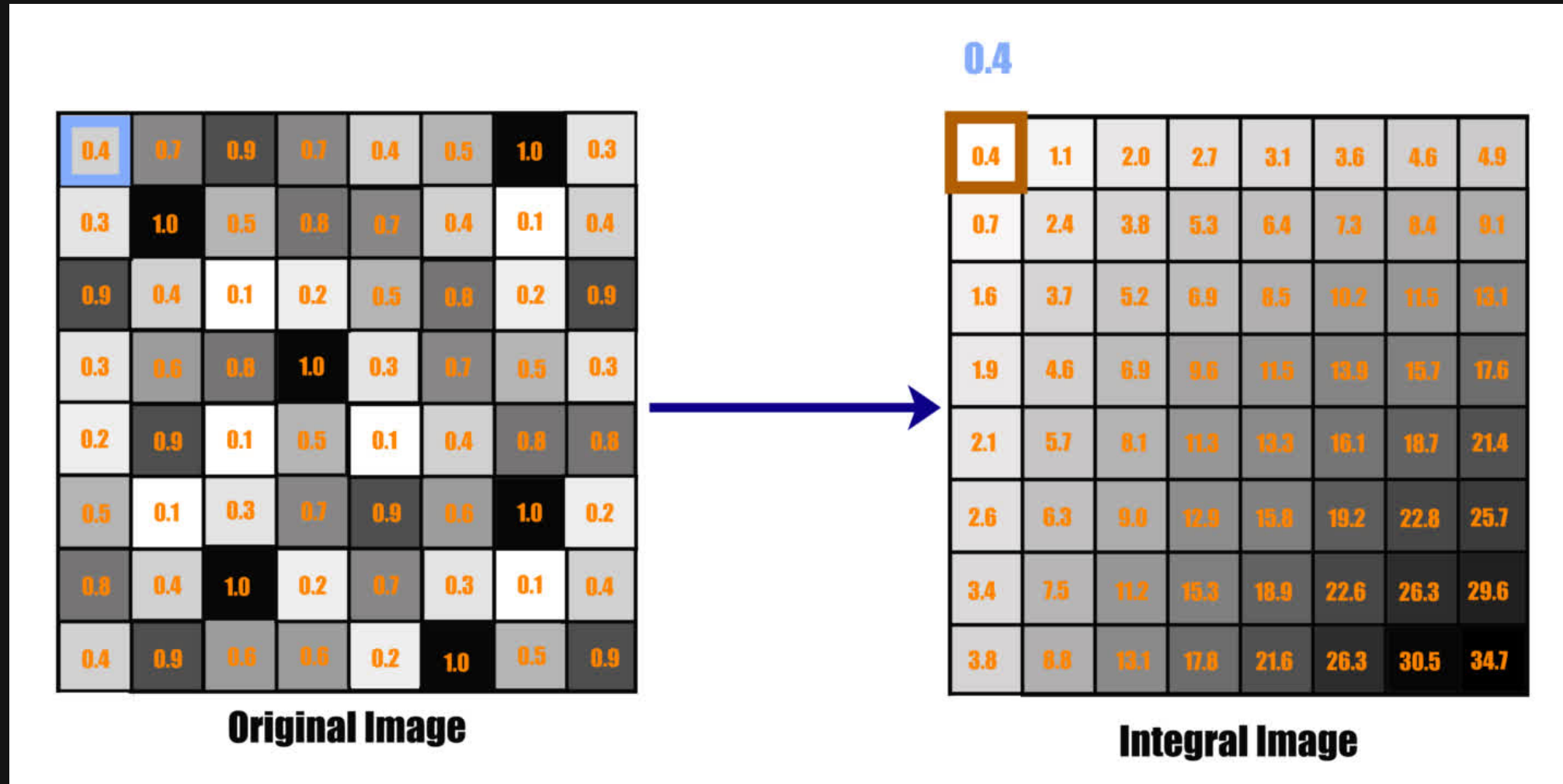
Many Haar features are passed over an image during training and we narrow down upon some features which have maximum ability of detecting a feature.



Is there a faster and simpler method  
to calculate the value??



# The Integral Image



An integral image is a modified representation of the original image wherein each pixel value is replaced by its sum with all the pixel values that lie to its left and top.

Thus, if we consider any grid of pixels in an image, the sum of all the pixel values is given by the rightmost and bottommost pixel value in the integral image.



# CALCULATING HAAR VALUE USING INTEGRAL IMAGE

0.4	0.7	0.9	0.7	0.4	0.5	1.0	0.3
0.3	1.0	0.5	0.8	0.7	0.4	0.1	0.4
0.9	0.4	0.1	0.2	0.5	0.8	0.2	0.9
0.3	0.8	0.8	1.0	0.3	0.7	0.5	0.3
0.2	0.9	0.1	0.5	0.1	0.4	0.8	0.8
0.5	0.1	0.3	0.7	0.9	0.8	1.0	0.2
0.8	0.4	1.0	0.2	0.7	0.3	0.1	0.4
0.4	0.9	0.8	0.8	0.2	1.0	0.5	0.9

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

1	0.4	1.1	2.0	2.7	3.1	3.6	4.6	4.9
2	0.7	2.4	3.8	5.3	6.4	7.3	8.4	9.1
3	1.6	3.7	5.2	6.9	8.5	10.2	11.8	13.1
4	1.9	4.6	6.9	9.9	11.8	13.9	15.7	17.6
5	2.1	5.7	8.1	11.3	13.3	16.1	18.7	21.4
6	2.6	6.3	9.0	12.8	15.8	19.2	22.8	25.7
7	3.4	7.5	11.2	15.3	18.9	22.6	26.3	29.6
8	3.8	8.8	13.1	17.8	21.6	26.3	30.5	34.7

SUM OF THE PIXELS IN DARK AREA/NUMBER OF PIXELS

=  $(26.3 - 15.3 - 4.6 + 2.7)/18 = 9.1/18 = 0.51$

1 2 3 4 5 6 7 8

0.4	1.1	2.0	2.7	3.1	3.6	4.6	4.9
0.7	2.4	3.8	5.3	6.4	7.3	8.4	9.1
1.6	3.7	5.2	6.9	8.5	10.2	11.8	13.1
1.9	4.6	6.9	9.9	11.8	13.9	15.7	17.6
2.1	5.7	8.1	11.3	13.3	16.1	18.7	21.4
2.6	6.3	9.0	12.8	15.8	19.2	22.8	25.7
3.4	7.5	11.2	15.3	18.9	22.6	26.3	29.6
3.8	8.8	13.1	17.8	21.6	26.3	30.5	34.7

SUM OF THE PIXELS IN LIGHT AREA/NUMBER OF PIXELS

=  $(15.3 - 3.4 - 2.7 + 0.4)/18 = 9.6/18 = 0.53$

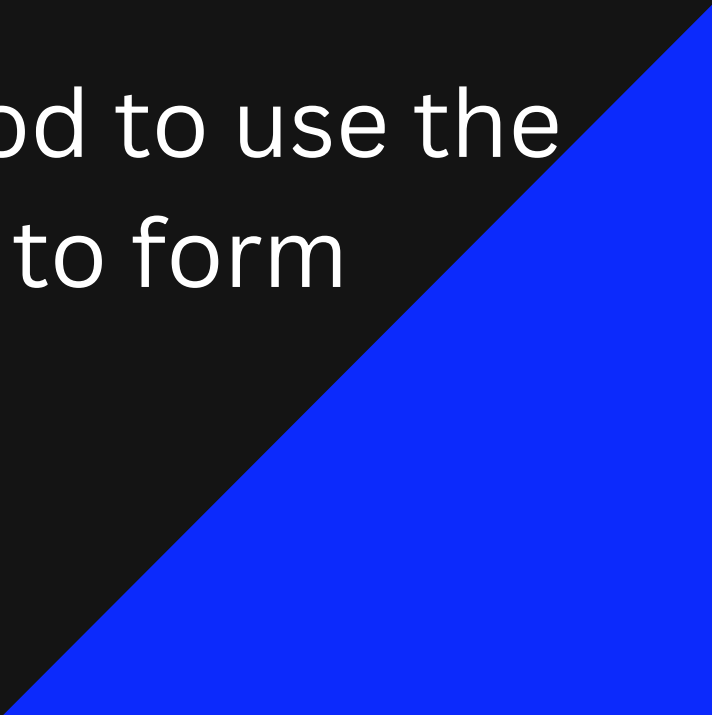
Haar Value  
 $0.51 - 0.53 = -0.02$

Can we make life easier??

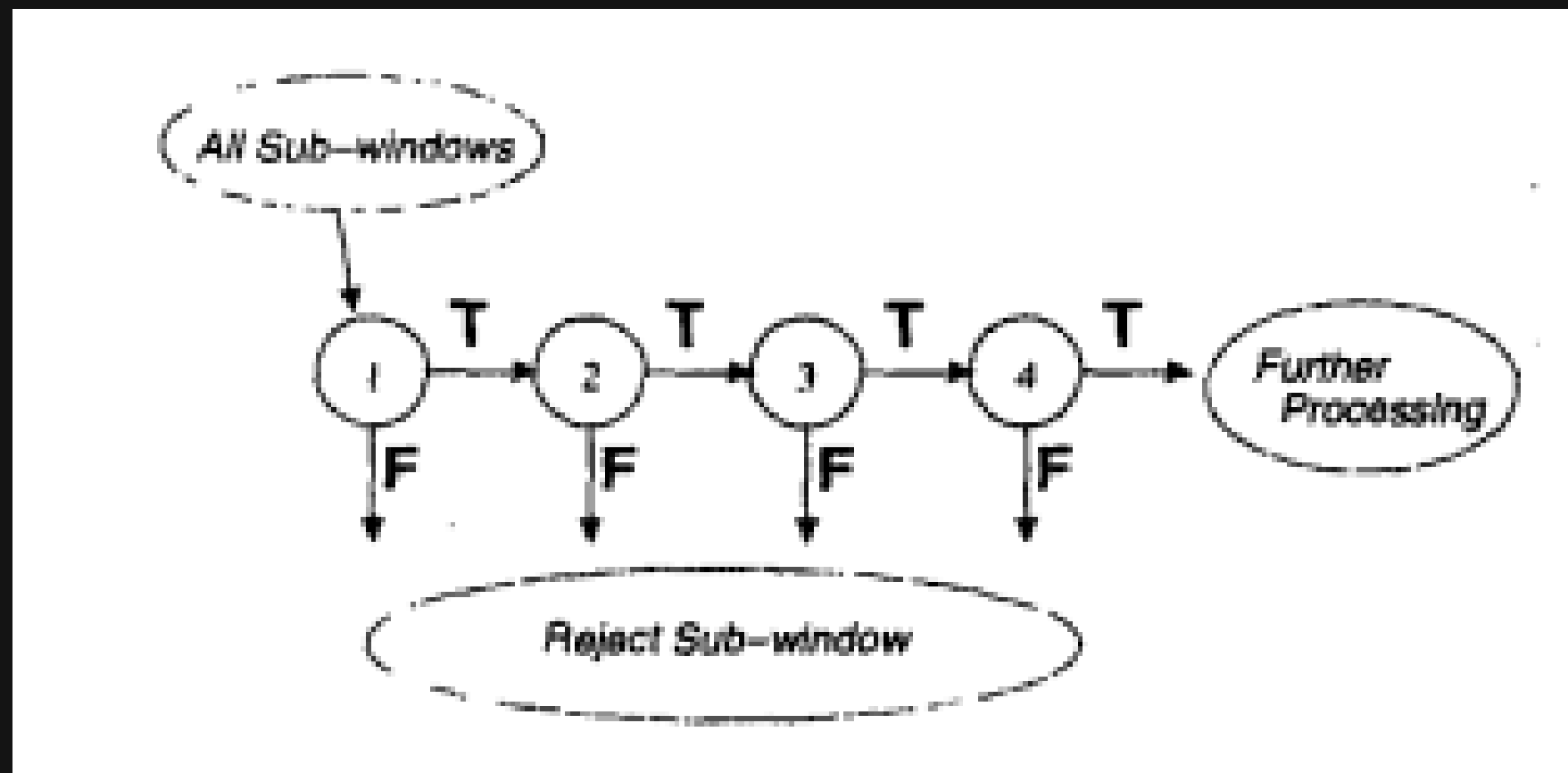




# AdaBoost

- AdaBoost performs the crucial task of creating a subset of features from a huge set of Haar features.
  - The features are applied separately through small models, resulting in less accuracy as all features are naturally not helpful.
  - This results in weak models also called 'Weak Learners' which are able to classify images with less accuracy.
  - AdaBoost then uses its ensemble learning method to use the results of the weak learners and combines them to form a strong model.
- 

# THE ATTENTION CASCADE




In the Attention Cascade, the image is passed through a series of Haar features in a sequence.

The features are aligned such that if a negative output is obtained from one feature, the image is discarded.



Let's take a short quiz to  
test our understanding!





# Histogram of Oriented Gradients

# Feature Descriptor

Feature Descriptors represent the image in a simplified format by extracting all the useful information and discarding extraneous information.

A feature vector converts an image of  $H \times W \times 3$  into a feature vector which is nothing but an array of numbers holding information represented by the image.

The HoG is also a feature descriptor which converts an image into a feature vector that has 3780 terms.



In an HoG, the distribution (histogram) of the direction of the gradients (oriented gradients) is stored in the feature vector. Note that the HoG algorithm does not classify the objects in the image directly. It is just a technique that represents the image more effectively and is used for further processing.

# What is a Gradient??

For functions involving multiple variables, gradient is a vector representation of the partial derivatives of the function.

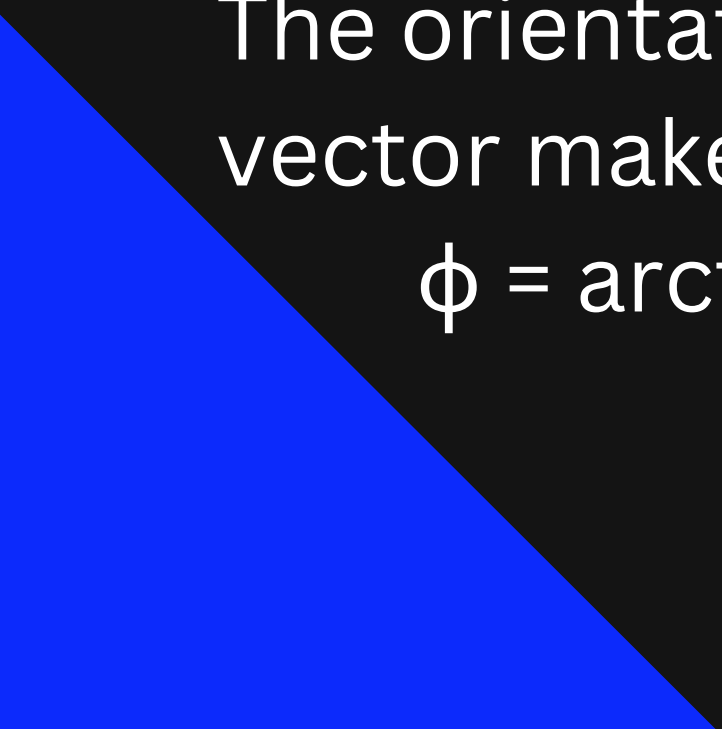
Thus, if we have a function  $f$  of independent variables  $x$  and  $y$ , the gradient is the vector  $(G_x, G_y)$  where  $G_x$  and  $G_y$  represent the partial derivatives of  $f$  w.r.t  $x$  and  $y$  respectively.

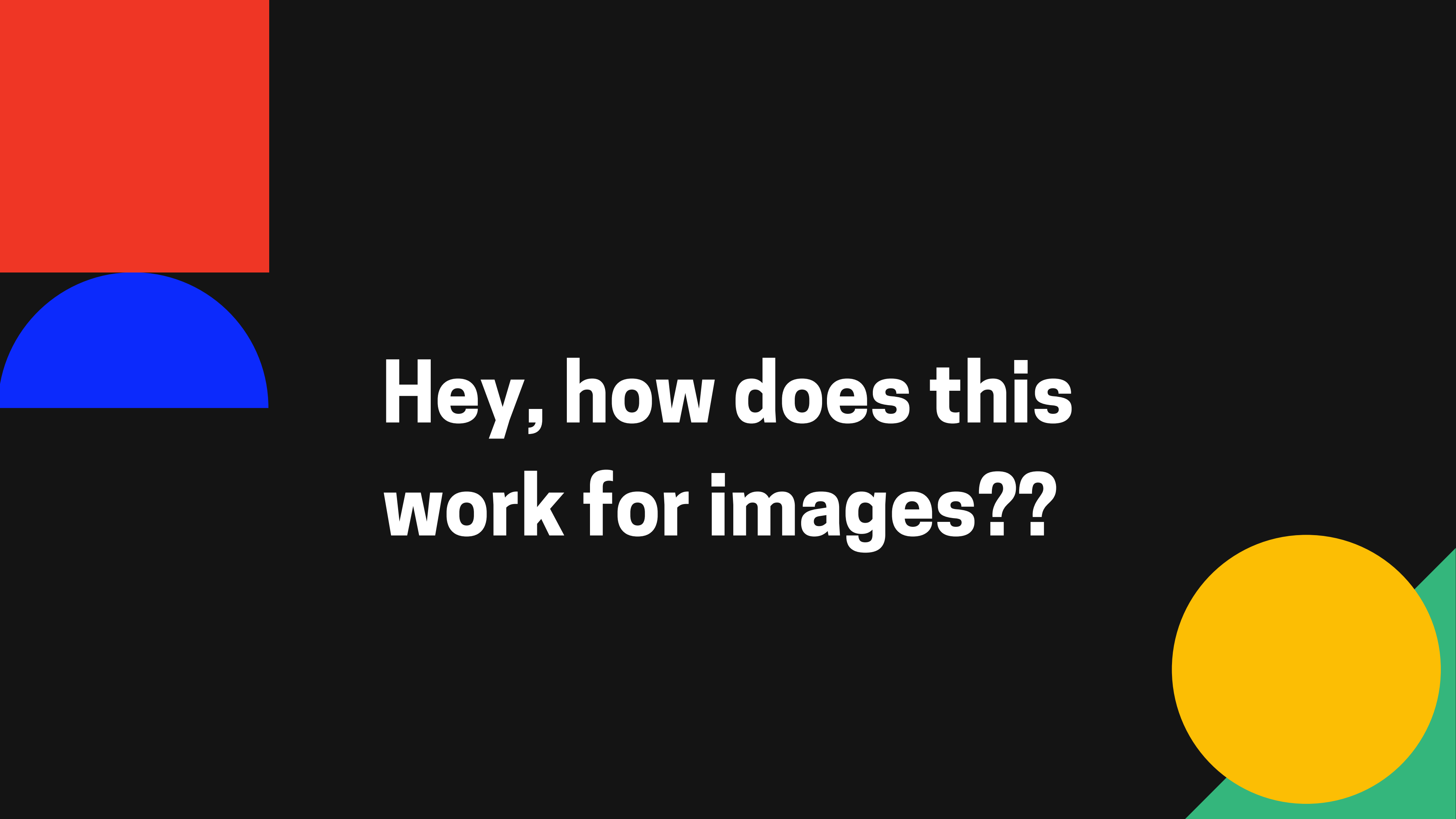
As the gradient is a vector, the magnitude of the gradient is given by

$$|G| = \sqrt{(G_x)^2 + (G_y)^2}$$

The orientation of the gradient i.e the angle that the gradient vector makes with the positive  $x$  axis is given by

$$\phi = \arctan(G_y/G_x)$$





**Hey, how does this  
work for images??**

$$G_x(r, c) = I(r, c + 1) - I(r, c - 1) \quad G_y(r, c) = I(r - 1, c) - I(r + 1, c)$$

where r, c refer to rows and columns respectively. (Image by author)

As per the given formula, for the cell with pixel value 85, the gradient in the x and y directions are -

$$G_x = 89 - 78 = 11$$

$$G_y = 68 - 56 = 12$$

Subsequently, the magnitude and angle are calculated for all pixel values and stored in 2 different arrays.

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

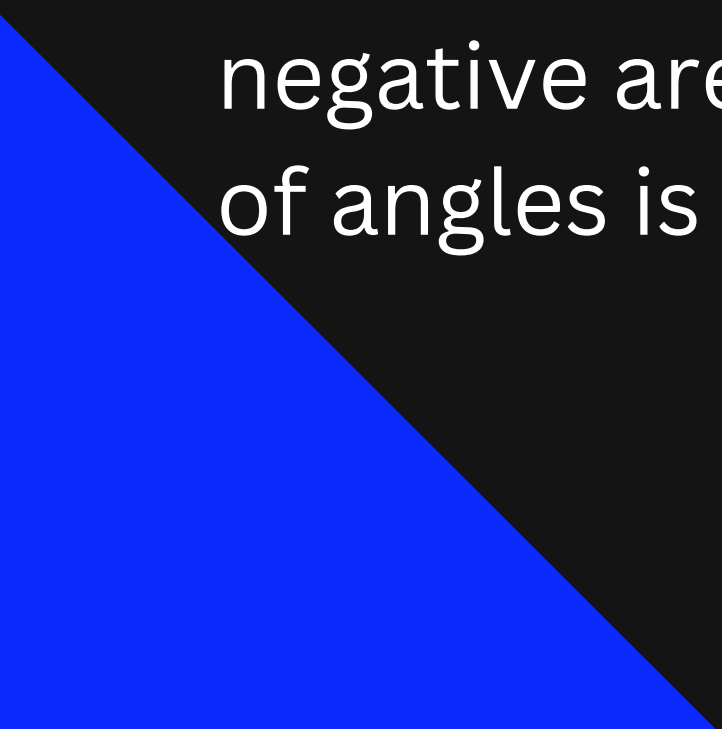


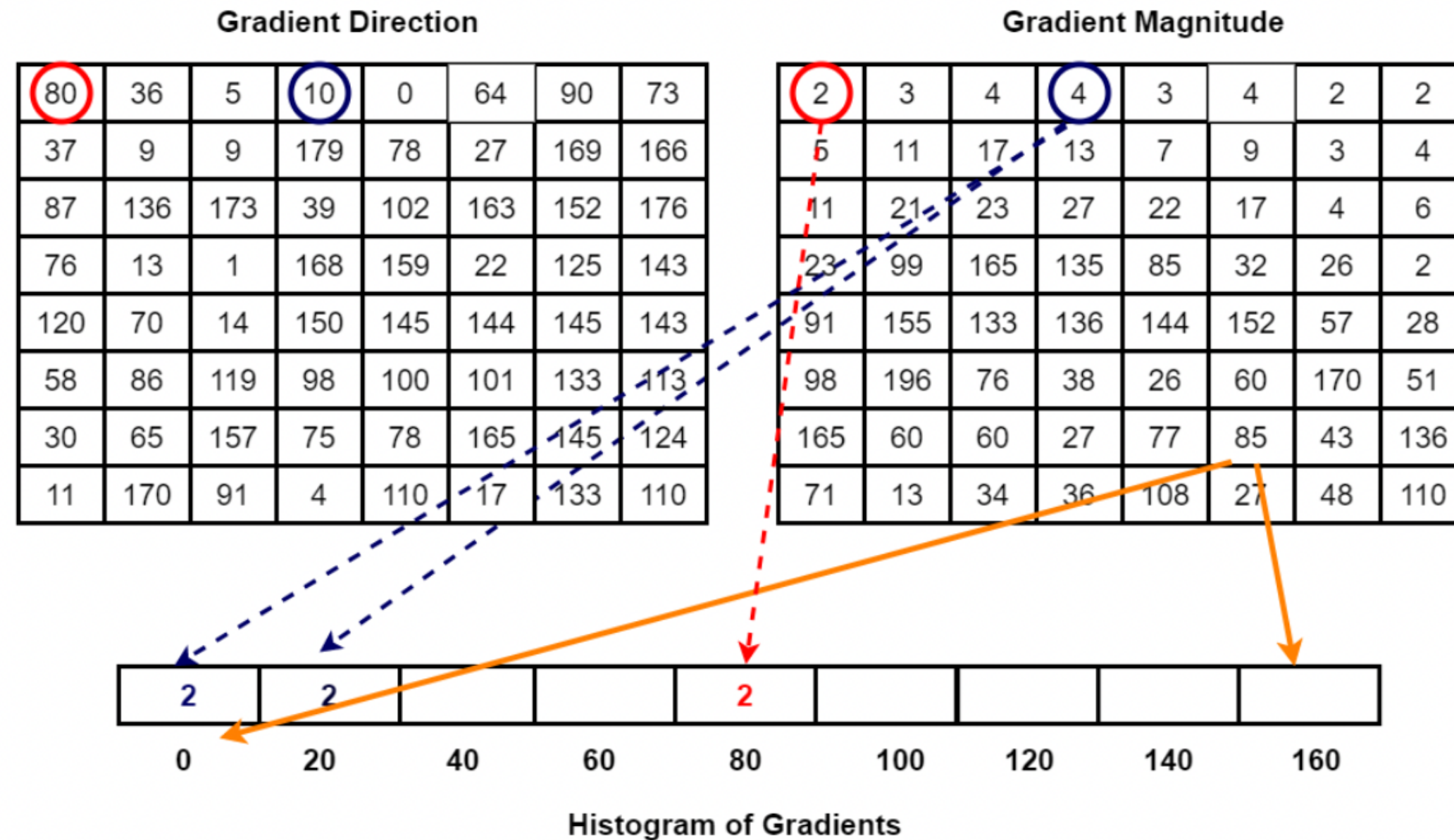
# Binning the Gradients

After obtaining the gradient values, the pixels are grouped in 8x8 matrices and for each matrix and distributed in 9 bins with a step size (gap) of 20 degrees.

$$\begin{aligned} \text{Number of bins} &= 9 (\text{ranging from } 0^\circ \text{ to } 180^\circ) \\ \text{Step size } (\Delta\theta) &= 180^\circ / \text{Number of bins} = 20^\circ \end{aligned}$$

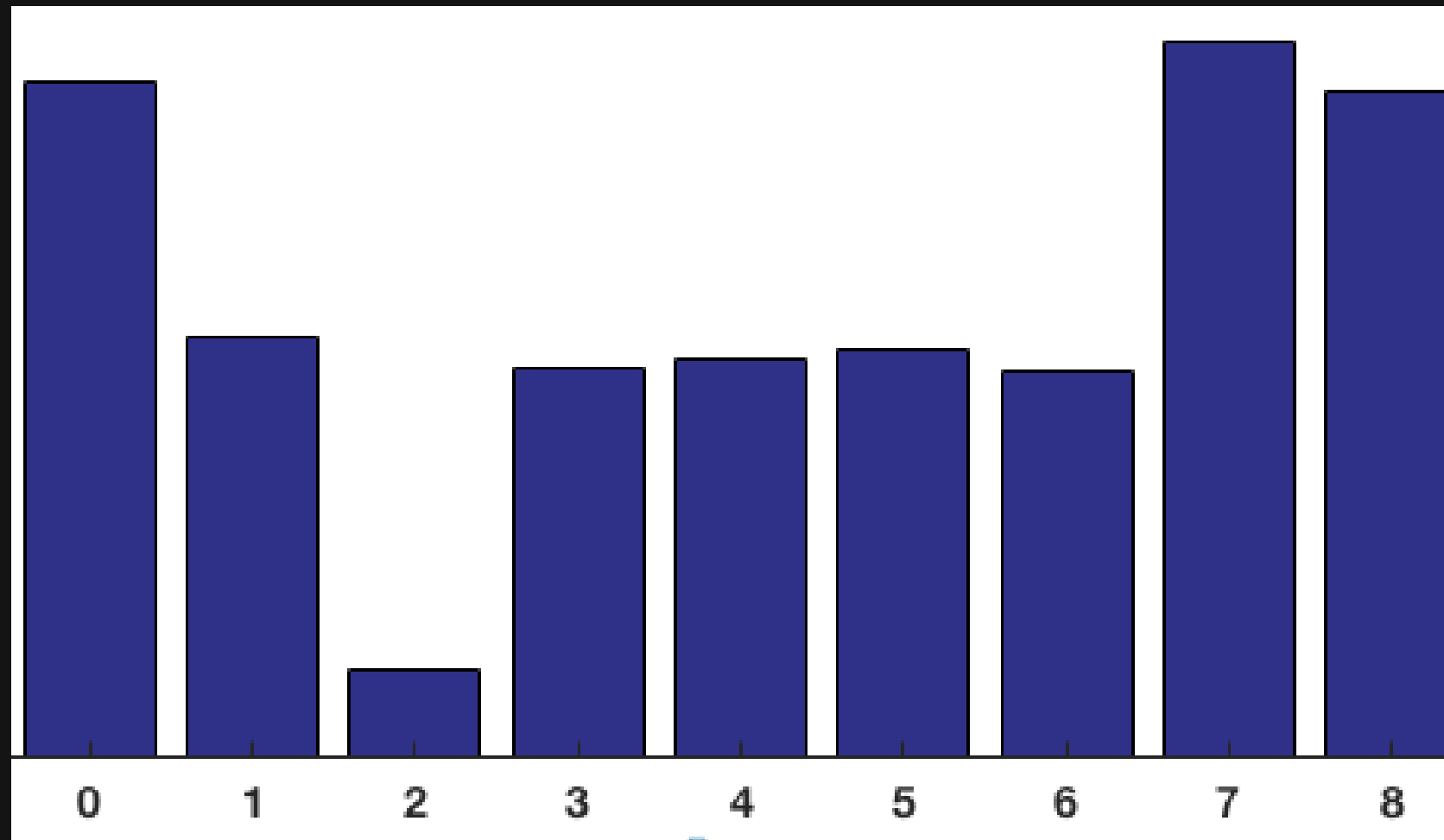
Note that the gradient values are unsigned, i.e the gradient and its negative are represented by the same value and hence, the range of angles is from 0 to 180 degrees and not 0 to 360.





If the values of gradient direction are integral multiples of 20, the corresponding magnitude is stored in the respective bins. However, when the direction is not an integral multiple, a weighted value of the magnitude is stored.

The weight basically is decided by how close the gradient directions is to its nearest multiples of 20.



The values are further normalised by dividing each value by the overall magnitude of the feature vector. It is mainly done to reduce the variations in gradient values.

Thus, we finally obtain the histogram representation of the gradient magnitudes.

An 8x8 matrix of pixels is thus converted into a 1x9 matrix of values. Further, such 4 8x8 matrices are grouped to form a 2x2 block i.e a large 16x16 matrix and correspondingly, it is converted into 1x36 array of values.

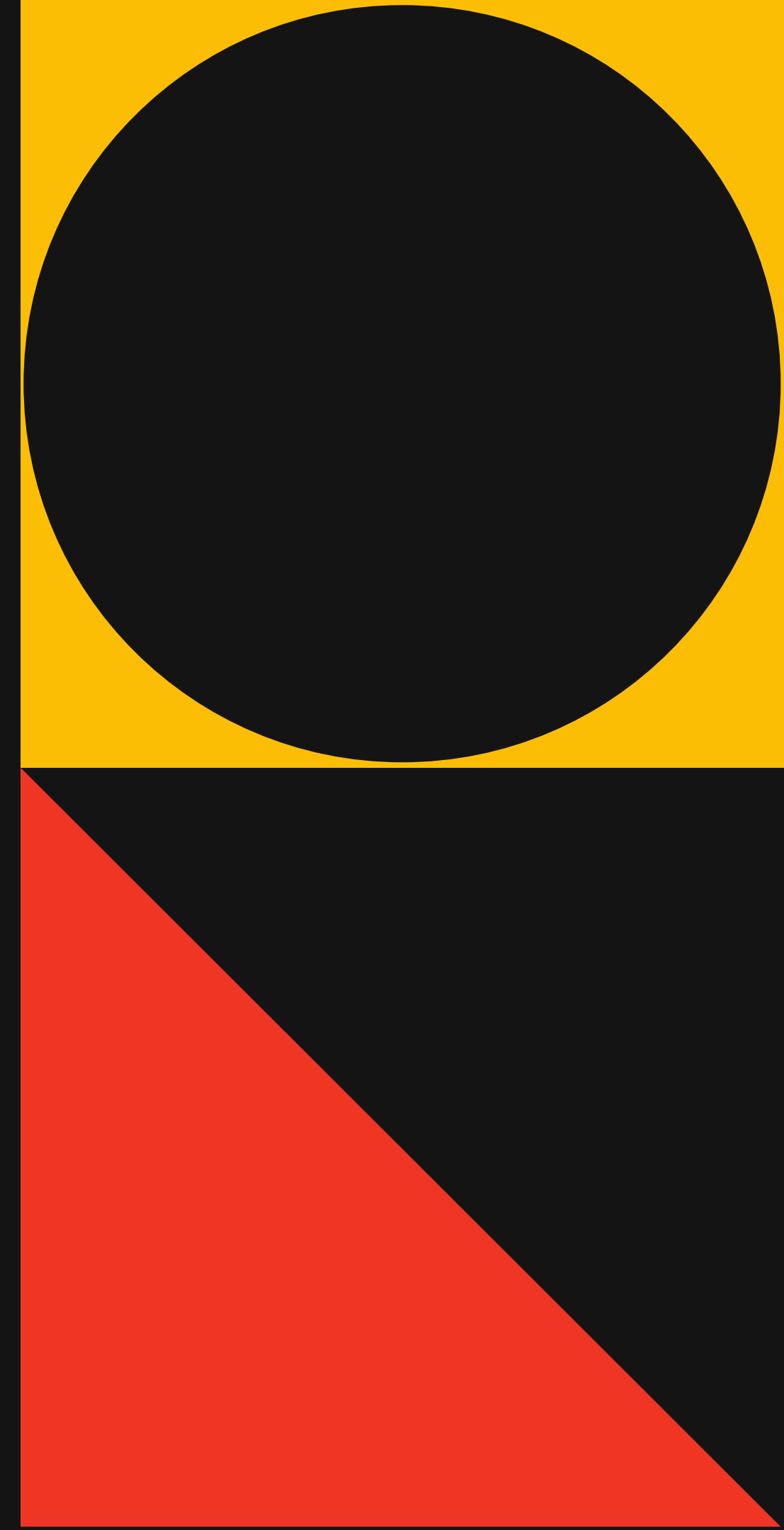
$$k = \sqrt{b_1^2 + b_2^2 + b_3^2 + \dots + b_{36}^2}$$
$$f_{bi} = \left[ \left( \frac{b_1}{k} \right), \left( \frac{b_2}{k} \right), \left( \frac{b_3}{k} \right), \dots, \left( \frac{b_{36}}{k} \right) \right]$$



Thus if we use a 64x128 image as shown, we obtain 8 8x8 grids along the width and 16 along the length.

Hence, we would require 7 16x16 blocks along the width and 15 8x8 blocks along length, resulting in  $15 \times 7 = 105$  feature vectors.

Each 16x16 feature vector stores 36 values and thus we obtain  $36 \times 105 = 3780$  total values stored by the HoG feature descriptor.





**We'll now have another small**  
**quiz and then move to code**  
**implementation**

