

# COMPUTER VISION

RECOGNITION WITH GLOBAL FEATURES

Ngo Quoc Viet-2024

# CONTENTS

---

1. Global features
  - Appearance based recognition
  - Image Histogram
  - Global Representation for Image Classification
2. Using global features for recognition
  - Histogram Backprojection
  - Multi-dimension histogram
  - Colored derivatives
3. Project:

# LECTURE OUTCOMES

- Understanding color histogram (including multi-dimension histogram), color derivatives and their applications.
- Implementing histogram backprojection to focus the specific object in an image or video.
- Using color histogram to compare the images and implementing the simplest classifier for images.
- Implementing the lecture project to detect brands in image/video.



# IMAGE FORMATION: BASICS

- Image  $f(x,y)$  is characterized by 2 components
  - Illumination  $i(x,y)$  = Amount of source illumination incident on scene
  - Reflectance  $r(x,y)$  = Amount of illumination reflected by objects in the scene

$$f(x,y) = i(x,y)r(x,y)$$

where  $0 < i(x,y) < \infty$  and  $0 \leq r(x,y) \leq 1$

$r(x,y)$  depends on object properties.  $r = 0$  means total absorption and 1 means total reflectance.

- For slowly varying illumination,  $i(x,y)$  will be the same at neighboring locations
- Ratio of neighboring locations is independent of illumination

$$\frac{f(x,y)}{f(x+1,y)} = \frac{i(x,y)r(x,y)}{i(x,y)r(x+1,y)} = \frac{r(x,y)}{r(x+1,y)}$$

- Taking logarithms:  $f(x,y) - \ln f(x+1,y) = \ln r(x,y) - \ln r(x+1,y)$ . This is just the derivative of the logarithm of image.

# APPEARANCE BASED RECOGNITION

- Appearance-based image recognition is a branch of computer vision that focuses on identifying objects or patterns in images based on their **visual appearance** (visual features of an object rather than its structural or semantic characteristics).
- Appearance-based methods are commonly used in various applications, including object recognition, image classification, and facial recognition.
- Some concepts and techniques associated with appearance-based image recognition: *Histogram, Feature Extraction* (HOG, SIFT); *Image Descriptor Representations* of the extracted features that can be used for matching and recognition; *Image Matching* to compare and find similarities between different images. Matching methods can include template matching, nearest-neighbor search, and machine learning-based approaches.

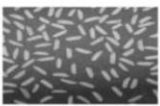

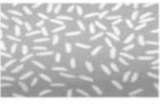
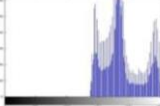



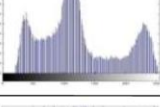

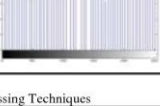
# GLOBAL FEATURES

- Global features capture information about the overall structure and content of an entire image, as opposed to local features that focus on specific regions or keypoints
- **Color Histograms:** Represents the distribution of colors in an image.
- **Texture Features:** Describes the overall texture pattern in an image such as Haralick texture features, Gabor filters, and Local Binary Patterns (LBP).
- **Shape Descriptors:** Represents the overall shape of objects in an image such as Hu Moments, Zernike Moments, and Fourier Descriptors.
- **Global Binary Patterns:** Represents the spatial arrangement of pixel values in a binary form such as LBP computed globally for the entire image.
- **Edge Histograms:** Represents the distribution of edge orientations in an image. Captures information about the overall structure of edges.

# IMAGE HISTOGRAM

- A histogram is a graph that shows frequency of anything.
- Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels intensity values.
- Histograms has many uses in image processing
  - Analysis of the image (draft prediction, Its like looking an x ray of a bone of a body).
  - Enhancement purposes (contrast, brightness).
  - Thresholding (in computer vision).

**Table 1:** Results of histogram processing techniques

Type	Image and its Histogram	
Input Image		
Histogram Sliding (Right Sliding)		
Histogram Sliding (Left Sliding)		
Histogram Stretching		
Histogram Equalization		

**TABLE2:** Comparison of Histogram Processing Techniques

Processing Technique	Fully Enhanced image	User Interactive	Complex	Histogram Shape Affected	Full Dynamic Range	Characteristics Affected
Histogram Sliding	No	Yes	No	No	Accordingly user input	Brightness
Histogram Stretching	Yes	Yes	No	No	Accordingly user input	Contrast
Histogram Equalization	Yes	No	Yes	Yes	Yes	Contrast

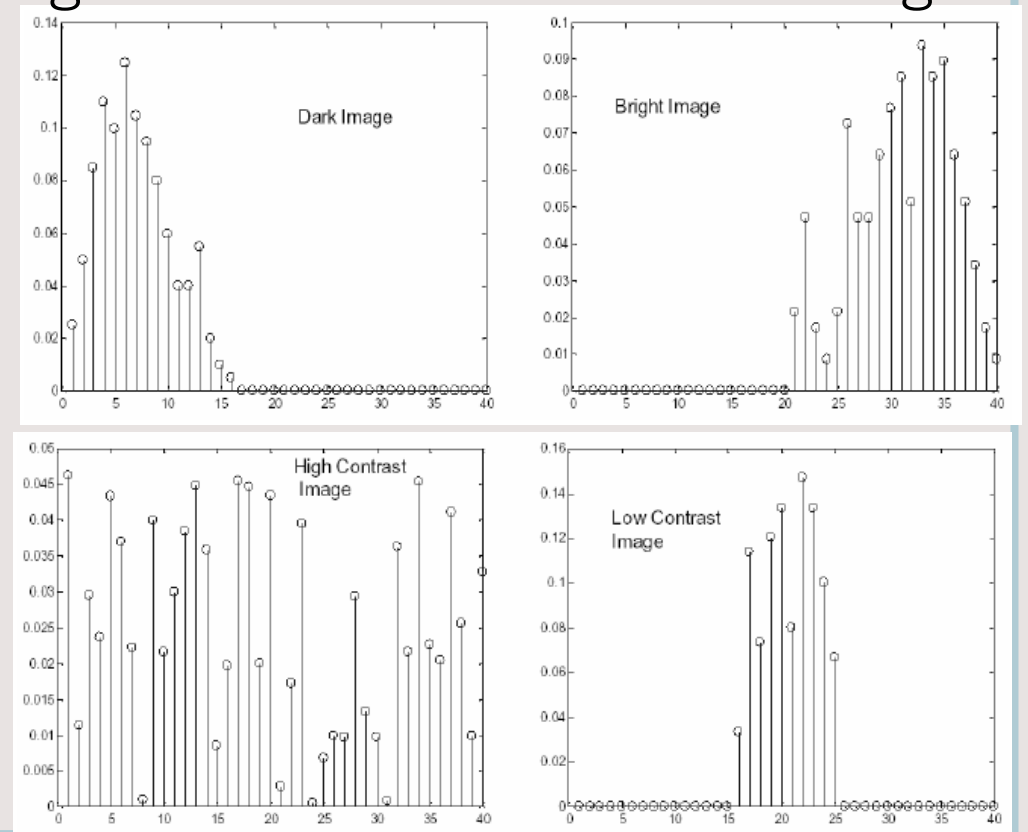
# IMAGE HISTOGRAM

- The gray level frequency  $g$  of image  $I$  is the number of pixels with value  $g$ . Histogram is a chart of gray levels in an image. For example, for image  $I$ , the histogram  $h(g)$  of  $I$  is

$$I = \begin{pmatrix} 1 & 2 & 0 & 4 \\ 1 & 0 & 0 & 7 \\ 2 & 2 & 1 & 0 \\ 4 & 1 & 2 & 1 \\ 2 & 0 & 1 & 1 \end{pmatrix}$$

$g$	0	1	2	4	7
$h(g)$	5	7	5	2	1

The histogram shape represents the brightness and contrast of the image





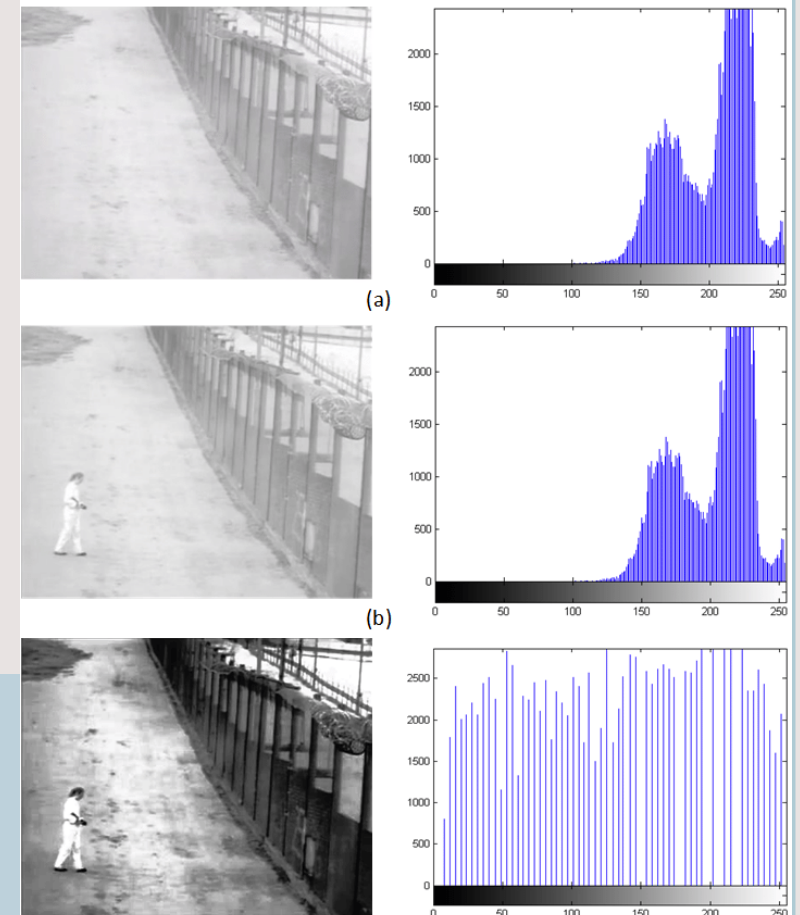
# HISTOGRAM EQUALIZATION

- Why histogram equalization?
  - Low contrast image: narrow luminance range.
  - Under-exposed image: concentrating on the dark side.
  - Over-exposed image: concentrating on the bright side.
- **Unbalanced** histogram do not fully utilize the dynamic range
- **Balanced** histogram gives more pleasant look and reveals rich details.

# Histograms Equalization of a image using cv2.equalizeHist()

```
cv.calcHist(img, [0, 1], None, [0, 256], [0, 256])
```

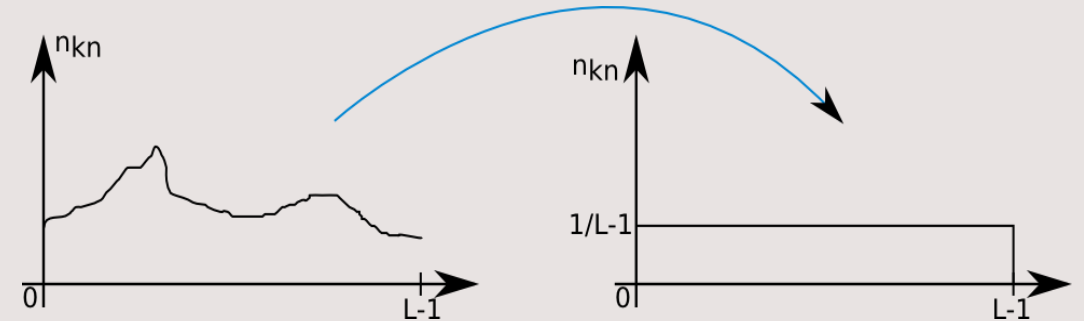
```
equ = cv2.equalizeHist(img)
```



Source: Researchgate

# OBJECTIVES OF HISTOGRAM EQUALIZATION

- Change the gray intensity of each pixel to get a new image with balanced histogram.
- Map the each luminance level to a new value such that the output image has approximately **uniform distribution** of gray levels.
- Two requirements
  - Monotonic (non-decreasing) function: no value reversals.
  - The output range being the same as the input range.
- How? Use **cumulative probability distribution (CDF)** function to equalize.
- Histogram equalization is very useful for scientific images like thermal, satellite or x-ray images.

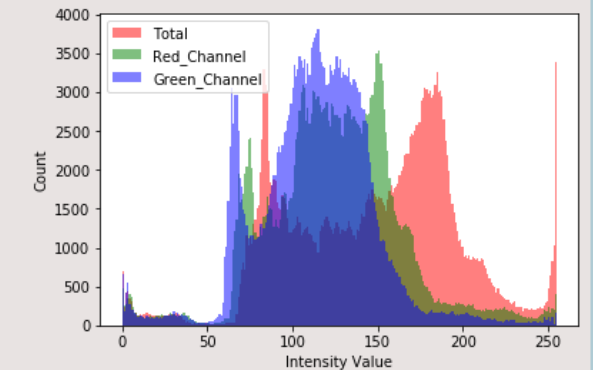
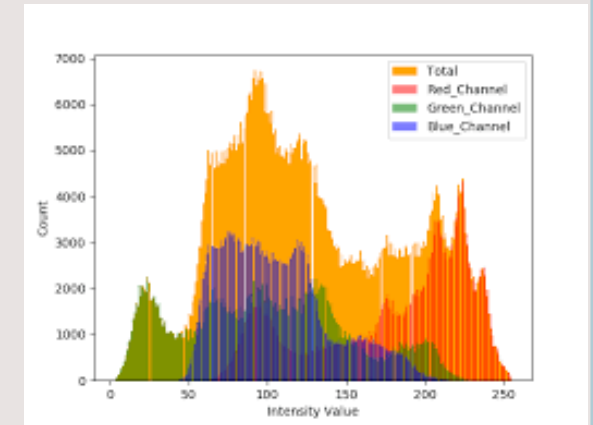


$$n_{kn} = \frac{n_k}{length \times width} = p_r(r_k)$$

$$\begin{cases} [0, L-1] \rightarrow N \\ x \rightarrow Card(x) \end{cases} \Rightarrow \begin{cases} [0, L-1] \rightarrow [0, 1] \\ x \rightarrow pdf(x) = \frac{Card(x)}{\sum_{i=0}^{L-1} Card(x_i)} \end{cases}$$

# MULTIDIMENSION HISTOGRAM

- Multidimensional histogram is a representation that captures the distribution of pixel values in an image across multiple dimensions (multiple features or channels simultaneously). For example, in an RGB color space, you might create a 3D histogram where the three axes correspond to the intensity values in the red, green, and blue channels.
- Multidimensional histograms are the extension of the concept of Image Histogram to Multi Channel images
- Multidimensional histograms are often used as image descriptors in computer vision tasks. For example, Histogram of Oriented Gradients (HOG) can be considered a type of multidimensional histogram (Lecture 5). But Haar-like, SIFT are not represented as a traditional multidimensional histogram.



# MULTIDIMENSION HISTOGRAM

- Multidimensional histograms are often used as image descriptors in computer vision tasks (HOG, Lecture 5).
- Multidimensional histograms can be used to compare the similarity between images based on their color or texture content.

```
def calculate2DHistogram(image):  
    # Convert the image to HSV color space  
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)  
    # Calculate 2D histogram for the H and S channels  
    hist = cv2.calcHist([hsv_image], [0, 1], None, [180, 256], [0, 180, 0, 256])  
    # Normalize the histogram  
    hist = cv2.normalize(hist, hist, 0, 1, cv2.NORM_MINMAX)  
    return hist
```

```
def compareHistograms(hist1, hist2, method=cv2.HISTCMP_CORREL):  
    # Compare histograms using a specified method (default is correlation)  
    similarity = cv2.compareHist(hist1, hist2, method)  
    return similarity
```

# MULTIDIMENSION HISTOGRAM – DISTANCE METRIC

- **HISTCMP\_CORREL**: Measures the correlation between two histograms. A value of 1 indicates a perfect match, while a value of -1 indicates a complete mismatch.

$$d(H_1, H_2) = \frac{\sum_i (H_1(i) - \overline{H_1})(H_2(i) - \overline{H_2})}{\sqrt{\sum_i (H_1(i) - \overline{H_1})^2 \sum_i (H_2(i) - \overline{H_2})^2}}$$

- **HISTCMP\_CHISQR** measure of the difference between the expected and observed frequencies of a set of categorical data. It is commonly used to quantify the dissimilarity between two histograms, a smaller value indicates a closer match.

$$d(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i)}$$

# MULTIDIMENSION HISTOGRAM – DISTANCE METRIC

- HISTCMP\_INTERSECT: Computes the intersection (minimum) between two histograms. A larger value indicates a closer match.

$$d(H_1, H_2) = \sum_i \min(H_1(i), H_2(i))$$

- HISTCMP\_BHATTACHARYYA: is often used to quantify the dissimilarity between two probability distributions, such as histograms. Values closer to 0 indicate a closer match.

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\overline{H_1 H_2 B^2} \sum_i \sqrt{H_1(i) \cdot H_2(i)}}}}$$

- HISTCMP\_WEMD is used to quantify the dissimilarity between two histograms, which measures the minimum cost of transforming one distribution into another.

# HISTOGRAM COMPARISON-THE SIMPLEST CLASSIFIER

- Calculate the histograms of the two input images using `cv2.calcHist()`
- Normalize the histograms computed above for the two input images using `cv2.normalize()`
- Compare these normalized histograms using `cv2.compareHist()`. It returns the comparison metric value. Pass suitable histogram comparison method as parameter to this method.
- For the Correlation and Intersection methods, the higher the metric, the more accurate the match. While for chi-square and Bhattacharyya, the lower metric value represents a more accurate match.

# HISTOGRAM COMPARISON-THE SIMPLEST CLASSIFIER

```
def calcHistogram(frame, colorSpace = 'HSV',
                  channels = [0, 1, 2], bins = [16, 16, 16], range=[0, 256, 0, 256, 0, 256]):
    if (colorSpace == 'HSV'):
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    elif (colorSpace == 'GRAY'):
        model_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    elif (colorSpace == 'RGB'):
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    hist = cv2.calcHist([frame], channels, None, bins, range)
    hist = cv2.normalize(hist, hist).flatten()

    return hist

def histogramComparision(h1, h2, method='intersection'):
    if (method == "intersection"):
        comparison = cv2.compareHist(h1, h2, cv2.HISTCMP_INTERSECT)
    elif (method == "correlation"):
        comparison = cv2.compareHist(h1, h2, cv2.HISTCMP_CORREL)
    elif (method == "chisqr"):
        comparison = cv2.compareHist(h1, h2, cv2.HISTCMP_CHISQR)
    elif (method == "battacharyya"):
        comparison = cv2.compareHist(h1, h2, cv2.HISTCMP_BHATTACHARYYA)
    else:
        raise ValueError(
            'The method specified ' + str(method) + ' is not supported.')
    return comparison
```



# ADVANTAGES OF COLOR HISTOGRAMS

- Fast and easy to compute
- Invariant to Translation, Rotation about the viewing axis
- Changes slowly with
  - Rotation about other axes (viewpoint changes)
  - Distance to object (scaling)
- Can be used for image matching in content-based image retrieval, object identification, and object location
- NOT invariant to changes in lighting conditions

## COLORED DERIVATIVES

- Colored derivatives refer to the partial derivatives of an image with respect to its color channels. These derivatives are used to analyze the rate of change of pixel intensities along different directions in the image. It is often used to enhance the edges of an image or a frame in video.
- Colored derivatives are often used in tasks like edge detection, texture analysis, and feature extraction in computer vision applications.
- Local gradient magnitude indicates edge strength.

$$|Grad(f(x, y))| = \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2} \approx \left|\frac{\partial f(x, y)}{\partial x}\right| + \left|\frac{\partial f(x, y)}{\partial y}\right|$$

# GRADIENT-BASED EDGE DETECTION NOTATIONS

- Gradient

$$\text{Grad}(f(x, y)) = \left[ \frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right]$$

- Gradient magnitude

$$|\text{Grad}(f(x, y))| = \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2} \approx \left| \frac{\partial f(x, y)}{\partial x} \right| + \left| \frac{\partial f(x, y)}{\partial y} \right|$$

- Gradient orientation

$$\theta = \tan^{-1} \left( \frac{\partial f(x, y)}{\partial y} / \frac{\partial f(x, y)}{\partial x} \right)$$

# GRADIENT FILTERS

Operator	Matrix form
Central Diffrence	$\begin{bmatrix} 0 & 0 & 0 \\ -1 & [0] & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ 0 & [0] & 0 \\ 0 & 1 & 0 \end{bmatrix}$
Roberts	$\begin{bmatrix} [0] & 1 \\ -1 & 0 \end{bmatrix} \quad \begin{bmatrix} [0] & 1 \\ -1 & 0 \end{bmatrix}$
Prewitt	$H_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & [0] & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \cdot [1 \quad 1 \quad 1] \quad H_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & [0] & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot [-1 \quad 0 \quad 1]$
Sobel	$H_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & [0] & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \cdot [1 \quad 2 \quad 1] \quad H_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & [0] & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot [-1 \quad 0 \quad 1]$
Orientation Sobel	$H_y = \begin{bmatrix} 0 & 1 & 2 \\ -1 & [0] & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad H_x = \begin{bmatrix} 0 & -1 & -2 \\ 1 & [0] & -1 \\ 2 & 1 & 0 \end{bmatrix}$

# COLORED DERIVATIVES AS GLOBAL FEATURES

```
# Function to compute colored derivatives using  
Sobel operator
```

```
def compute_colored_derivatives(image):
```

```
    # Convert the image to the LAB color space
```

```
    lab_image = color.rgb2lab(image)
```

```
    # Compute derivatives for each LAB channel
```

```
    sobel_dx = np.zeros_like(lab_image)
```

```
    sobel_dy = np.zeros_like(lab_image)
```

```
    for i in range(3): # Process each LAB channel
```

```
        sobel_dx[:, :, i] = sobel(lab_image[:, :, i],  
axis=0)
```

```
        sobel_dy[:, :, i] = sobel(lab_image[:, :, i],  
axis=1)
```

```
    return sobel_dx, sobel_dy
```

```
image = io.imread("image.jpg")
```

```
sobel_dx,          sobel_dy          =
```

```
compute_colored_derivatives(image)
```

```
magnitude      =      np.sqrt(sobel_dx**2      +  
sobel_dy**2)
```

```
feature_vector          =
```

```
np.concatenate([sobel_dx.flatten(),  
sobel_dy.flatten()])
```

```
#Load images and convert them by using  
feature_vector
```

```
svm_classifier = SVC()
```

```
svm_classifier.fit(X_train, y_train)
```

```
predictions = svm_classifier.predict(X_test)
```

# HISTOGRAM BACK PROJECTION

- It is used for image segmentation or **finding objects** of interest in an image. It is used with object tracking algorithms like **meanshift** and **camshift**.
- It was proposed by Michael J. Swain, Dana H. Ballard in their paper *Indexing via color histograms*, Third international conference on computer vision, 1990. This was one of the first works that use color to address the classic problem of Classification and Localisation in computer vision.
- Histogram Backprojection answers the question “*Where are the colors in the image that belong to the object being looked for (the target)?*”
- Histogram Backprojection involves the use of histograms and a backward projection process

# HISTOGRAM BACK PROJECTION

- **Histogram** is used to represent the distribution of pixel intensities in an image.
- **Back Projection** involves projecting this histogram information back onto the image to highlight regions that match a given histogram.
- The name “Histogram back projection” essentially describes a process where a histogram is used to identify and highlight regions in an image that have a similar tonal distribution to a specified reference histogram.
- This technique is often employed in object detection and tracking, where the goal is to find regions in an image that resemble a certain target based on their pixel intensity distribution.

# HISTOGRAM BACK PROJECTION-ALGORITHM

1. Convert images (object/ROI and target) into HSV and calculate their histograms
  - $M$  = histogram of ROI;  $I$  = histogram of target image.

```
roi = cv2.imread('../datasets/roi.jpg')
hsvr = cv2.cvtColor(roi,cv2.COLOR_BGR2HSV)
target = cv2.imread('../datasets/target.jpg')
hsvt = cv2.cvtColor(target,cv2.COLOR_BGR2HSV)
M = cv2.calcHist([hsvr],[0, 1], None, [180, 256], [0, 180, 0, 256])
I = cv2.calcHist([hsvt],[0, 1], None, [180, 256], [0, 180, 0, 256])
```

2. Find the ratio  $R = M/I$

$$R = M/(I+1)$$



# HISTOGRAM BACK PROJECTION-ALGORITHM

3. Backproject R, ie use R as palette and create a new image with every pixel as its corresponding probability of being target.  $B(x,y) = R[h(x,y),s(x,y)]$  where h is hue and s is saturation of the pixel at (x,y). After that apply the condition  $B(x,y) = \min[B(x,y), 1]$ .

```
h,s,v = cv2.split(hsvt)
```

```
B = R[h.ravel(),s.ravel()]
```

```
B = np.minimum(B,1)
```

```
B = B.reshape(hsvt.shape[:2])
```

4. Fine-tuning B by applying a convolution with a circular disc,  $B = D * B$ , where D is the disc kernel.

```
disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
```

```
cv2.filter2D(B,-1,disc,B)
```

```
B = np.uint8(B)
```

```
cv2.normalize(B,B,0,255,cv2.NORM_MINMAX)
```

# HISTOGRAM BACK PROJECTION-ALGORITHM

5. Now the location of maximum intensity gives us the location of object. If we are expecting a region in the image, thresholding for a suitable value gives a nice result.

```
ret,thresh = cv2.threshold(B,10,255,0)  
# Overlay images using bitwise_and  
thresh = cv2.merge((thresh,thresh,thresh))  
res = cv2.bitwise_and(target,thresh)
```

# HISTOGRAM BACK PROJECTION IN OPENCV

- OpenCV provides an inbuilt function:

*`cv2.calcBackProject( target_img, channels, roi_hist, ranges, scale )`*

*# calculating object histogram*

*`M = cv2.calcHist([hsvr],[0, 1], None, [180, 256], [0, 180, 0, 256])`*

*# normalize histogram and apply backprojection*

*`cv2.normalize(M,M,0,255,cv2.NORM_MINMAX)`*

*`B = cv2.calcBackProject([hsvt],[0,1],M,[0,180,0,256],1)`*

# ASSIGNMENT-HISTOGRAM BACKPROJECTION

- Using histogram backprojection to locate for blue striped shirt in the image.
  - Rotate the target image to 15, 30, 45 degrees and re-run the test and show the results.
  - Scaling the target image to 1.1, 1.2, 1.5, etc, and re-run the test and show the results.
- Turn in: resulted screens+code screen.



Pic from:Swain & Ballard

# PROJECT: BRAND RECOGNITION

- Problem 1: using histogram backprojection to detect one logo in an image
  - Using subset of brand logo dataset (1079 files).  
<https://www.kaggle.com/datasets/momotabanerjee/brand-logo-recognition-dataset>.
- Problem 2: logo recognition
  - Using small subset of Logo-2k-plus-Dataset:  
[https://drive.google.com/drive/folders/1PTA24UTZcsnzXPN1gmV0\\_IRg3IMHqwp6](https://drive.google.com/drive/folders/1PTA24UTZcsnzXPN1gmV0_IRg3IMHqwp6).
- Problem 3: logo detection in video

Root Category	Logos	Images
Food	769	54,507
Clothes	286	20,413
Institution	238	17,103
Accessories	210	14,569
Transportation	203	14,719



# PROJECT: BRAND RECOGNITION-INSTRUCTIONS

- Convert images to YCbCr color space (cv2.COLOR\_RGB2YCrCb).

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \frac{1}{256} \begin{bmatrix} 65.738 & 120.057 & 25.064 \\ -37.945 & -74.494 & 112.439 \\ 112.439 & -94.154 & -18.285 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- Using 32 bins histogram per axis. In order to providing invariance to intensity, we can use Y and Cr channels.