GNR638: Feature Extraction from Images

Detailed Mathematical Notes & Derivations

Your Name

January 12, 2026

Contents

Chapter 1

Vision Features

1.1 Image Features Vectors

Image feature vectors are encoding tool that help represent images in a way that is suitable for Machine Learning Tasks and Algorithms.
Mathematically, an image (or a patch of an image) is a high-dimensional object. If you have an image patch of size $N \times M$ pixels, raw data lives in $\mathbb{R}^{N \times M}$. A feature extraction function f maps this raw data into a more manageable, meaningful vector space $\mathbb{R}^d$ (where d is the feature dimension)

$$\mathbf{x} = f(\text{Image}) \in \mathbb{R}^d$$

The vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ is the feature vector. Each component x_i captures a specific characteristic of the image, such as color intensity, texture patterns, edge orientations, or more complex attributes learned through deep learning models.

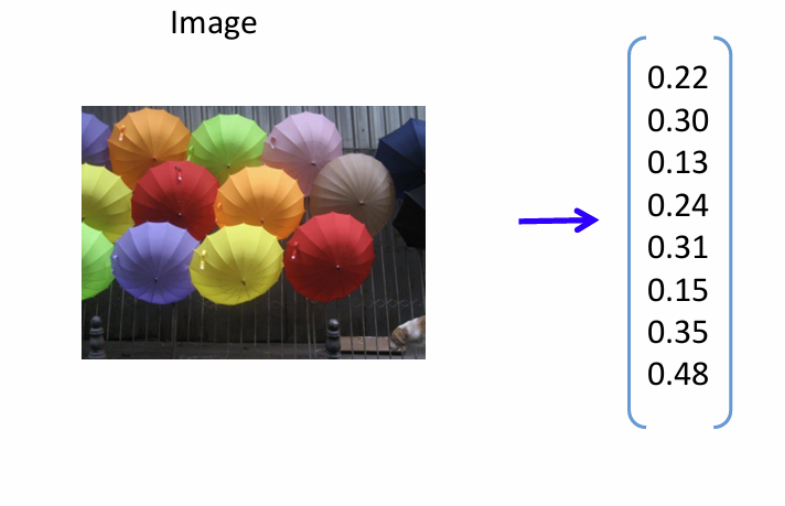


Figure 1.1: Block diagram of the system

1.1.1 Why are they Useful?

Feature vectors makes it possible for ML to work. Example we want to vectorise the images and then cluster them in the $\mathbb{R}$ space, such that within class variance is very low and between class variance is high.

- vectorisation ⇒ numerical representation of images
- clustering ⇒ grouping similar images together
- within class variance ⇒ how similar images in the same group are
- between class variance ⇒ how different images in different groups are

1.2 Visual Features

There are 4 types of visual features:

- Color Features
- Texture Features
- Shape Features
- Deep features

1.3 Color Features

1.3.1 Binning

Image → 3 color channels (R,G,B) → 0 to 255 (256) values → Total Combinations = 256^3 .
Too many combinations, so we reduce them by grouping the near values into Bins.
The Math of Binning:For a pixel intensity p, its bin index b is calculated as:

$$b = \left\lfloor \frac{p}{W} \right\rfloor$$

Example: If Pixel Value = 100 and Bin Width = 32.$b = \lfloor 100/32 \rfloor = \lfloor 3.125 \rfloor = 3$.So, pixel 100 falls into Bin 3.

1.3.2 Color Histogram

Creation of Histogram is just the number of pixels falling into a specific bin Mathematically the equation for histogram bin count is given by -

$$h[k] = \sum_{x=0}^{H-1} \sum_{y=0}^{W-1} \mathbb{I}\left(\left\lfloor \frac{I_c(x,y)}{\text{bin\_width}} \right\rfloor = k \right)$$

The Issue with color histogram is it is tied to the size of the image and therefore two images having the same color profile can have different lokking histograms if there is difference in size of the images to deal with this shastro mein Normalisation ka zikr hai !!
Normalisation of Histogram is done by dividing each bin count by the total number of pixels in the image. ⇒ histogram = probability distribution making it invariant to image size.

Figure 1.2: Block diagram of the system

1.3.3 Parametric Density Estimation

The Idea is to instead of storing the entire histogram as color feature we reduce the features to just Mean and Variance of the histogram or a Gaussian curve fitted over the Histogram Data. There are 3 methods in which we can do it.

Method 1: Per-Channel Statistics (Independent Gaussians) This is the simplest approach. We assume the Red, Green, and Blue channels are completely unrelated (independent). We simply ask: "What is the average Red?" and "How much does the Red vary?"The Math: For each channel $c \in \{R, G, B\}$, we compute two statistics over N pixels:Mean (μ_c): The average intensity.

$$\mu_c = \frac{1}{N} \sum_{i=1}^{N} x_{i,c}$$

Standard Deviation (σ_c): The spread/contrast.

$$\sigma_c = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{i,c} - \mu_c)^2}$$

The Feature Vector: We store these values for all 3 channels.

$$\mathbf{f} = [\mu_R, \sigma_R, \mu_G, \sigma_G, \mu_B, \sigma_B]$$

Dimension: 6 parameters2.Limitation: It destroys the relationship between colors. If an image has many Yellow pixels (High Red + High Green), this method just sees "High Red" and "High Green" independently. It doesn't know they occurred together in the same pixel.

Method 2: 3D Multivariate Gaussian (Single Gaussian) This method treats the color as a single 3D vector $\mathbf{x} = [R, G, B]^T$. It fits a 3D ellipsoid to the data cloud. This captures not just the spread, but the correlation between channels.The Math:Mean Vector ($\boldsymbol{\mu}$): A 3D vector representing the center of the color cloud.

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$$

Covariance Matrix (Σ): A 3×3 symmetric matrix capturing how channels move together.

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T$$

Understanding Covariance:The diagonal entries ($\Sigma_{11}, \Sigma_{22}, \Sigma_{33}$) are the variances of R, G, B (same as Method 1).The off-diagonal entries (e.g., Σ_{12}) tell us if Red and Green are correlated. If Σ_{RG} is high, it means the image contains colors like Yellow (R+G) or Cyan, rather than just random noise.The Feature Vector:3 values for $\boldsymbol{\mu}$.6 values for Σ (since it is symmetric, $\Sigma_{RG} = \Sigma_{GR}$, so we only need the unique upper-triangular elements)3.Dimension: $3 + 6 = \mathbf{9}$ parameters.

Method 3: Gaussian Mixture Models (GMM) The previous methods assume the image has only one dominant color cluster (unimodal). But what if the image has a red shirt, blue sky, and green grass? A single Gaussian would try to fit a giant blob in the middle (which would be gray), failing to capture the distinct colors.GMM fits K different Gaussians to the data simultaneously.The Math: We model the probability of observing a pixel color $\mathbf{x}$ as a weighted sum of K Gaussians:

$$\mathfrak{p}(\mathbf{x}) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \Sigma_k)$$

π_k (Mixing Coefficient): The "weight" or importance of the k-th Gaussian (e.g., "30 percent of the image belongs to the grass cluster"). $\sum \pi_k = 1.\boldsymbol{\mu}_k$: The center color of cluster $k.\Sigma_k$: The spread of cluster k.The Feature Vector:For every cluster k, we store its Weight (π), Mean (μ), and Covariance (Σ).Simplification: To save space, we often assume Σ_k is diagonal (ignoring correlations within the cluster) because the multiple clusters already handle the complex shape4.Dimension (assuming diagonal Σ):Per Gaussian: $1(\pi) + 3(\mu) + 3(\text{diag}(\Sigma)) = 7$ parameters.Total: $7 \times K$ parameters.

1.4 Texture

Texture :

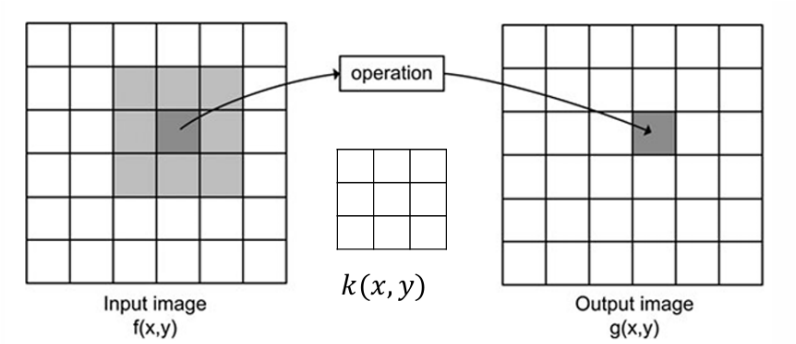
1.5 Image Filtering and Convolution

- Convolution and Image filtering are used to generate the words for bag of visual words. Convolution = Mathematical Engine of Computer Vision.

- Convolution takes tw functions to produce output, that corresponds to the amount of overlap between the two functions.

- image = first function, filter/kernel = second function.
- $f(x, y)$: The Input Image (intensity at x, y).
- $k(u, v)$: The Kernel (or Filter/Mask) of size $(2h + 1) \times (2w + 1)$.
- $g(x, y)$: The Output Image (Filtered).

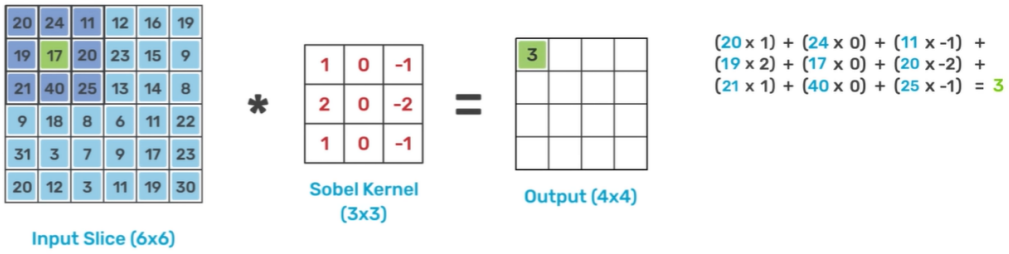
$$g(x, y) = (f * k)(x, y) = \sum_{v=-h}^{h}\sum_{u=-w}^{w} k(u, v)f(x - u, y - v)$$

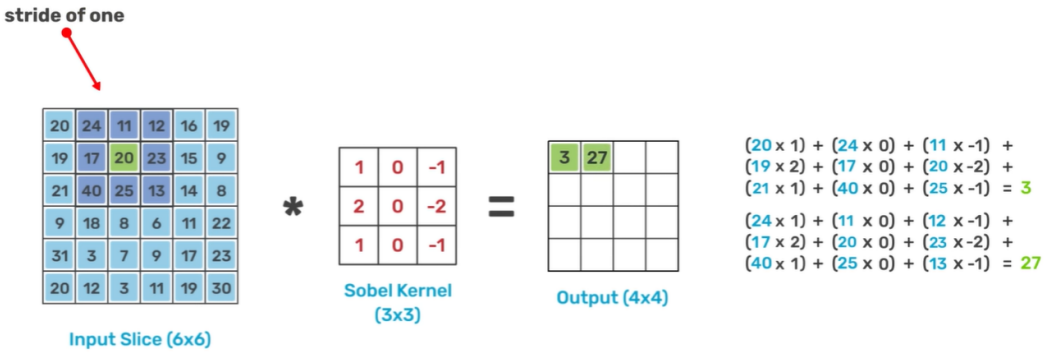


$$g(x,y) = \sum_v \sum_u k(u,v) f(x-u, y-v)$$

1.5.1 How Convolution Works?

Here is the illustration of convolution example:





1.5.2 Some Mathematical results

Convolving an image of size $H \times W$ with a filter (kernel) of size $h \times w$ (assuming h, w are odd numbers like 3, 5, 7) **Output Dimensions** are given by:

- Output Height: $H_{out} = H - h + 1$
- Output Width: $W_{out} = W - w + 1$

The above results are for No Padding and Stride of 1.
Padding is adding extra border pixels around the image.
Stride is the step size with which we slide the filter over the image.
With Padding P and Stride S, the output dimensions become:

- Output Height: $H_{out} = \frac{H-h+2P}{S} + 1$

- Output Width: $W_{out} = \frac{W-w+2P}{S} + 1$
- Note: Ensure that $(H - h + 2P)$ and $(W - w + 2P)$ are divisible by S for integer output dimensions.
- Example: For $H = 32, W = 32, h = 5, w = 5, P = 2, S = 1$:
- $H_{out} = \frac{32-5+2*2}{1} + 1 = 32$
- $W_{out} = \frac{32-5+2*2}{1} + 1 = 32$
- So, the output image remains 32×32.
- This is called "same" convolution because the output size is the same as the input size.

1.6 Edge Detection using Convolution

1.6.1 First order edge Detection

First order detectors use the first derivatives, which measures the rate of change of intensity. Mathematically, the gradient of the image intensity function $I(x, y)$ is given by:

$$\nabla F = \left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}\right]$$

where $\frac{\partial F}{\partial x}$ are given by the formula:

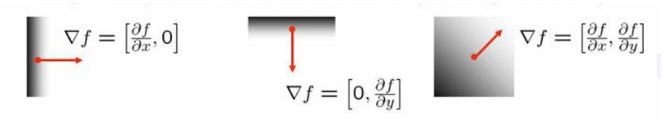
$$\frac{\partial F}{\partial x} = F(x + 1, y) - F(x, y)$$

and $\frac{\partial F}{\partial y}$ is given by the formula:

$$\frac{\partial F}{\partial y} = F(x, y + 1) - F(x, y)$$

The magnitude of the gradient vector gives the strength of the edge:

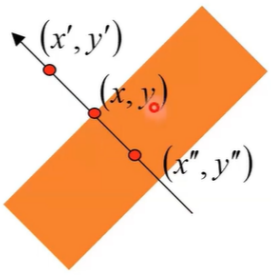
$$|\nabla F| = \sqrt{\left(\frac{\partial F}{\partial x}\right)^2 + \left(\frac{\partial F}{\partial y}\right)^2}$$



From the given figure it is evident that the direction of edge is perpendicular to the gradient direction. The direction θ of the edge can be calculated as:

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

After calculateting the gradient magnitude and direction, we can apply Non-maximum Suppression and Thresholding to get thin and clean edges.



$$M(x,y)=\begin{cases}|\nabla S|(x,y) & \text{if } |\nabla S|(x,y)>|\Delta S|(x',y') \\ & \&\,|\Delta S|(x,y)>|\Delta S|(x'',y'') \\ 0 & \text{otherwise}\end{cases}$$

1.6.2 NMS - Non-Maximum Suppression

Basically, it is a technique used to thin out the edges detected in an image by retaining only the local maxima in the gradient direction.

1.6.3 Histogram of Gradient (HOG-descriptor)

HoG is a feature descriptor that maps the weighted distribution of gradient values to the gradient oreintaion.

- Divide the image into small regions called cells (e.g., 8x8 pixels).
- For each cell, compute a histogram of gradient directions (e.g., 9 bins covering 0-180 degrees).
- Each pixel in the cell votes for a bin based on its gradient magnitude and orientation.
- Group adjacent cells into larger blocks (e.g., 2x2 cells).
- Normalize the histograms within each block to account for illumination variations.

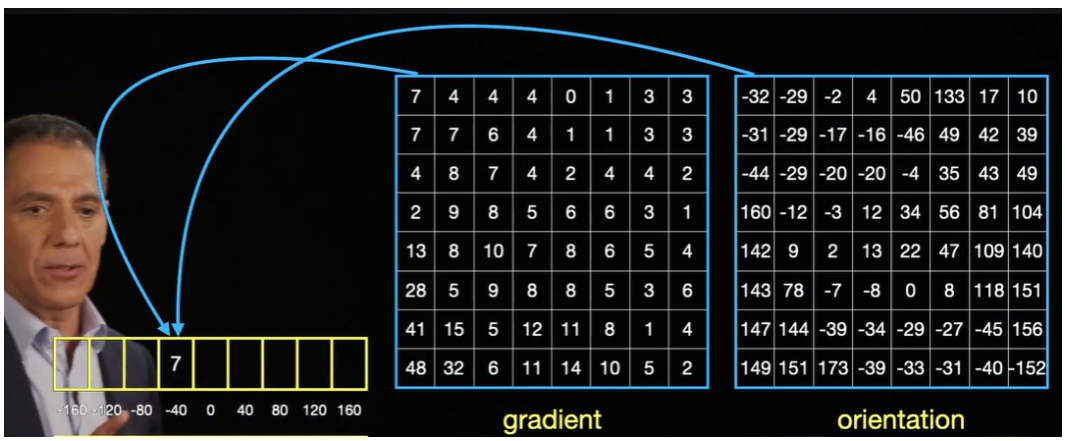


Figure 1.3: Histogram of Oriented Gradients (HOG) Descriptor

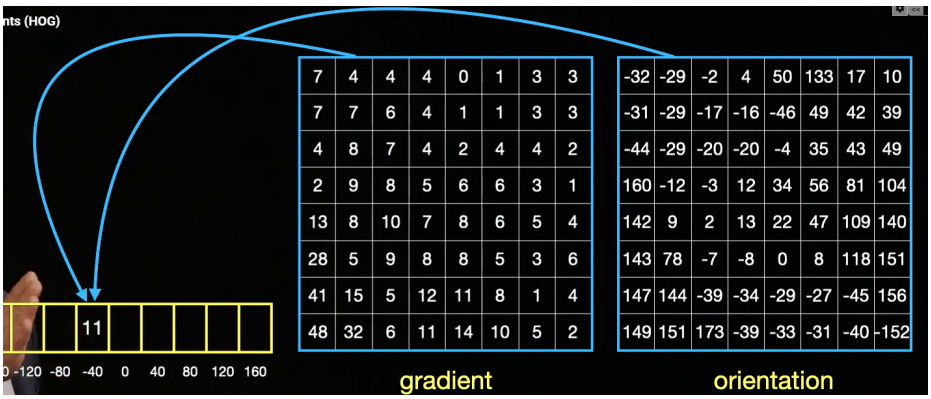


Figure 1.4: HOG Feature Visualization

Since HOG is obtained Patchwise we get a histogram for each of the patch and then we can plot the histograms as shown in the figure below:

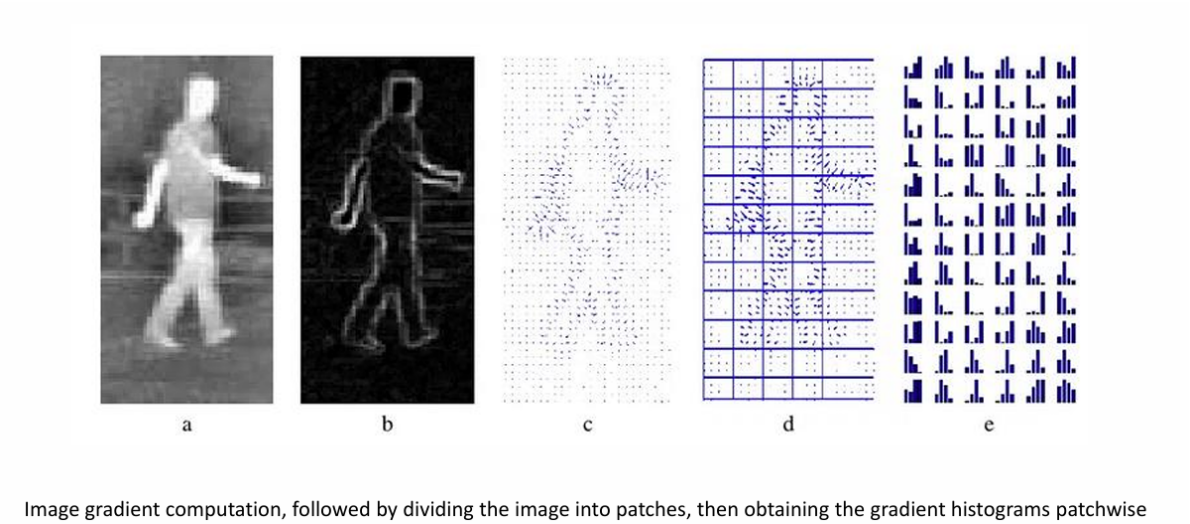


Figure 1.5: HOG Feature Visualization Patchwise

1.7 Points of Interest (Keypoints/Interest Points)

Mathematically, it is a point where the local image structure contains rich information. If you cut out a small patch around an interest point, that patch should look unique enough that you can find its exact twin in another image, even if the lighting or viewpoint has changed.

Three Catagories of points are there in an image:

- **Flat regions** are bad as once cut from the image there is no way to identify exact location.
- **Edge points** are better than flat regions but still not good enough as they can be identified only along one direction.
- **Corner points** are the best as they have high variation in all directions and therefore can be identified uniquely.

Examples of untility of Keypoints mentioned in the class are Oreientation detection and Paranoma Stiching. In both the cases we tend to track the keypoints in both the images and depending on the use either they are matched with each other or maped across each of the images.

1.8 Bag of Visual Words BoVW

"The" Image Encoding Technique

- Detect Keypoints.
- Take a patch sourrounding each keypoint.
- Describe them using HOG
- Now Use BoVW to encode the image.

Before we can describe an image, we need a common vocabulary. We build this by examining the distribution of features across the entire training dataset.1. Input Data PreparationLet $\mathcal{I}_{train} = \{I_1, I_2, \ldots, I_T\}$ be the set of T training images.From each image I_t, we extract a set of local HoG descriptors.Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ be the aggregate collection of all descriptors from all training images, where M is very large (millions).Each descriptor $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional vector (e.g., for standard HoG blocks, d might be 36 or 128).2. The K-Means Clustering AlgorithmWe use K-Means to group these millions of descriptors into K clusters. Each cluster center represents a "Visual Word."Goal: Find a set of K centroids $\mathcal{C} = \{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K\}$ that minimizes the Within-Cluster Sum of Squares (WCSS).Objective Function:

$$J = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in S_k} ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$$

Where S_k is the set of descriptors assigned to cluster k.3. The Iterative OptimizationStep A (Assignment): Assign every descriptor $\mathbf{x}_i$ to the nearest centroid.

$$z_i^{(t)} = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k^{(t)}||_2^2$$

Step B (Update): Recalculate the centroids based on the new assignments.

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{1}{|S_k|} \sum_{\mathbf{x}_i \in S_k} \mathbf{x}_i$$

Convergence: Repeat until assignments z_i do not change.

Encoding & Pooling (Constructing the Histogram)Now, for a specific image (training or test), we want to convert its collection of descriptors into a single fixed-length vector.1. InputAn image I containing N local descriptors: $X_I = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$.Note: N varies from image to image, but we need a fixed output size.2. Vector Quantization (Hard Assignment)We map each continuous descriptor $\mathbf{x}_i$ to the index of the nearest visual word in our codebook $\mathcal{V}$. This is mathematically defined as a mapping function $q(\mathbf{x})$:

$$q(\mathbf{x}_i) = \arg\min_{k \in \{1,\ldots,K\}} ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2$$

This step discretizes the continuous feature space. A complex 128-D vector is replaced by a single integer k2222.3. Pooling (Histogram Construction)We aggregate these assignments into a histogram $\mathbf{h} \in \mathbb{R}^K$. This removes spatial information (we know what appeared, but not where).For each visual word k (from 1 to K), the histogram bin h_k is calculated as:

$$h_k = \sum_{i=1}^{N} \delta(q(\mathbf{x}_i) - k)$$

Where $\delta(\cdot)$ is the Dirac delta function (returns 1 if the argument is 0, else 0).In simple terms: Count how many descriptors in the image were assigned to Word k.4. NormalizationSince images have different numbers of features N (e.g., a high-res image has more patches than a low-res one), raw counts are not comparable. We must normalize the histogram to transform it into a probability distribution or unit vector.L1 Normalization (Frequency): Sum of elements becomes 1.

$$\mathbf{h}_{L1} = \frac{\mathbf{h}}{\sum_{j=1}^{K} h_j} = \frac{\mathbf{h}}{N}$$

Interpretation: "Visual Word k makes up 10percentage of the image features."L2 Normalization (Euclidean): Unit length vector (preferred for SVMs).

$$\mathbf{h}_{L2} = \frac{\mathbf{h}}{||\mathbf{h}||_2} = \frac{\mathbf{h}}{\sqrt{\sum_{j=1}^{K} h_j^2}}$$

1.8.1 Example: Constructing Feature Vectors (Mini-SIFT)

To understand the contents of a feature vector $\mathbf{x}_i$ and the dataset $\mathcal{X}$, consider a simplified "Mini-SIFT" descriptor with the following parameters:

- **Patch Size:** 4×4 pixels.
- **Grid:** 2×2 sub-regions (Quadrants: TL, TR, BL, BR).
- **Bins:** 2 orientation bins (Horizontal vs. Vertical).
- **Total Dimension:** 4 quadrants $\times$ 2 bins = 8.

Step 1: Constructing Vector $\mathbf{x}_1$ (Vertical Edge)

Consider a patch P_1 containing a vertical edge:

$$P_1 = \begin{bmatrix} 10 & 10 & 50 & 50 \\ 10 & 10 & 50 & 50 \\ 10 & 10 & 50 & 50 \\ 10 & 10 & 50 & 50 \end{bmatrix} \tag{1.1}$$

The gradients show strong horizontal change ($G_x \approx 40$) in the right half, and zero vertical change ($G_y = 0$).

Voting by Quadrant:

- **Top-Left (TL):** No gradients $\rightarrow [0, 0]$
- **Top-Right (TR):** Strong Horizontal $\rightarrow [80, 0]$
- **Bottom-Left (BL):** No gradients $\rightarrow [0, 0]$
- **Bottom-Right (BR):** Strong Horizontal $\rightarrow [80, 0]$

Concatenating these gives the feature vector $\mathbf{x}_1$:

$$\mathbf{x}_1 = [\underbrace{0, 0}_{\text{TL}}, \underbrace{80, 0}_{\text{TR}}, \underbrace{0, 0}_{\text{BL}}, \underbrace{80, 0}_{\text{BR}}] \tag{1.2}$$

Step 2: Constructing Vector $\mathbf{x}_2$ (Horizontal Edge)

Consider a second patch P_2 containing a horizontal edge:

$$P_2 = \begin{bmatrix} 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 \\ 50 & 50 & 50 & 50 \\ 50 & 50 & 50 & 50 \end{bmatrix} \tag{1.3}$$

Here, gradients are purely vertical ($G_y \approx 40$).

$$\mathbf{x}_2 = [\underbrace{0,0}_{\text{TL}}, \underbrace{0,0}_{\text{TR}}, \underbrace{0,80}_{\text{BL}}, \underbrace{0,80}_{\text{BR}}] \tag{1.4}$$

Step 3: The Data Matrix $\mathcal{X}$

The collection $\mathcal{X}$ is the stack of all local descriptors extracted from the training images. If our dataset only contained these two patches, $\mathcal{X}$ would be:

$$\mathcal{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 80 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 0 & 0 & 80 & 0 & 80 \end{bmatrix} \tag{1.5}$$

This matrix $\mathcal{X}$ (of size $M \times D$) is the input to the **K-Means** algorithm used to learn the Visual Dictionary.

1.8.2 Example: Encoding and Pooling

Step 4: The Visual Dictionary (Codebook)

After running K-Means on the dataset $\mathcal{X}$, assume we find $K = 2$ cluster centers (Visual Words) representing the canonical features:

- **Word 1 ($\boldsymbol{\mu}_1$):** Vertical Edge Prototype.
- **Word 2 ($\boldsymbol{\mu}_2$):** Horizontal Edge Prototype.

Step 5: Encoding a New Image

Consider a new query image from which we extract 3 patches. We compute the Euclidean distance from each patch descriptor to the dictionary centroids and assign it to the nearest word (Vector Quantization).

1. **Patch A:** Descriptor matches $\boldsymbol{\mu}_1$ perfectly.
   $\rightarrow$ Assigned to **Word 1**.
2. **Patch B:** Descriptor matches $\boldsymbol{\mu}_1$ perfectly.
   $\rightarrow$ Assigned to **Word 1**.
3. **Patch C:** Descriptor is close to $\boldsymbol{\mu}_2$ (Horizontal).
   $\rightarrow$ Assigned to **Word 2**.

Step 6: Pooling (Histogram Construction)

We aggregate the assignments into a histogram $\mathbf{h} \in \mathbb{R}^K$.

- Count for Word 1: 2
- Count for Word 2: 1

The raw histogram is $\mathbf{h}_{raw} = [2, 1]$. To make this invariant to the number of patches (image size), we apply L1 normalization (dividing by $N = 3$):

$$\mathbf{h}_{final} = \left[\frac{2}{3}, \frac{1}{3}\right] \approx [0.67, 0.33] \tag{1.6}$$

This final vector $[0.67, 0.33]$ is the **Bag of Visual Words** representation of the new image, which serves as the input to the final classifier (e.g., SVM).