GNR638: Feature Extraction from Images

Detailed Mathematical Notes & Derivations

Your Name

January 10, 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.

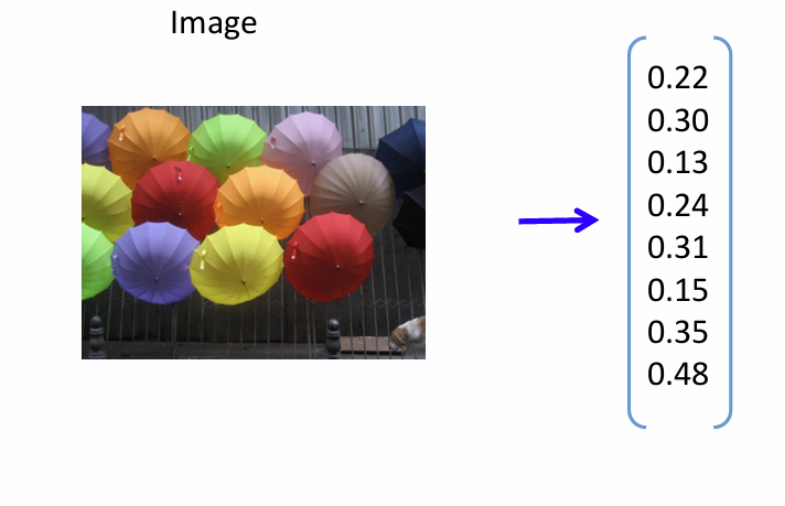


Figure 1.1: Block diagram of the system

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 :

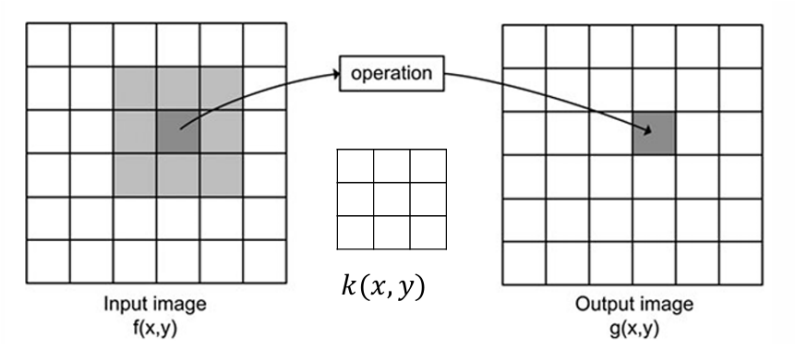
1.5 Image Filtering and Convolution

- 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).

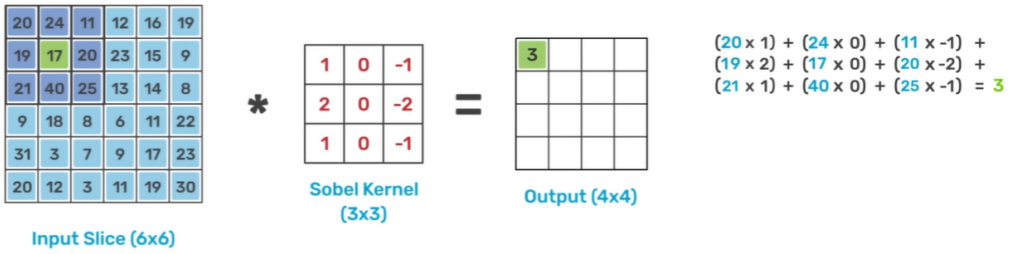
$$g(x, y) = (f * k)(x, y) = \sum_{v=-h}^{h}\sum_{u=-w}^{w} k(u, v)f(x - u, y - v)$$

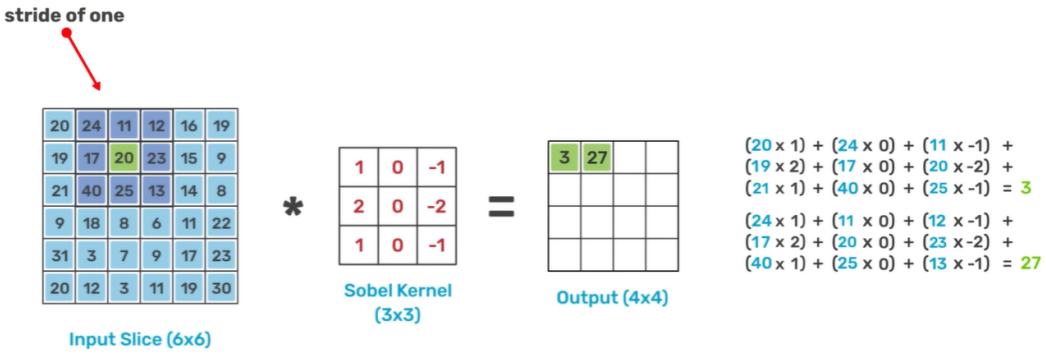


$$g(x,y) = \sum_v \sum_u k(u,v) f(x-u, y-v)$$

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:

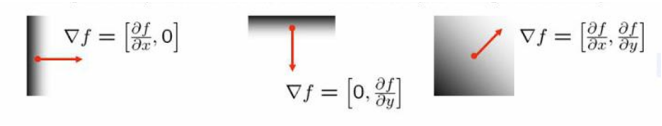
$$\frac{\partial F}{\partial x} = F(x + 1, y) - F(x, y)$$

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:

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



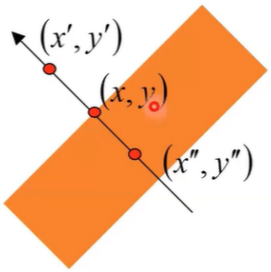
From the given figure it is evident that the direction of edge is perpendicular to the gradient direction. The direction θ of the edge can be calculated as:

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

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

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.



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