American International University -Bangladesh (AIUB)

Department of Computer Science & Engineering

Course Name: Machine Learning

Section: A

Report

Feature selection, extraction, and visualization techniques for image processing

Supervised By

Dr. Md. Asraf Ali

Submitted By

Name	ID
RIFAH SANZIDA	22-47154-1

Introduction

In image processing and computer vision, a feature represents an important piece of information extracted from an image, helping computers better understand and process images. Features highlight patterns, shapes, edges, colors, and textures, facilitating tasks such as object recognition, image classification, and motion tracking. By focusing on key attributes, features allow efficient image analysis without significant information loss.

Examples of Features

- Pixel Features: Brightness or color values (grayscale or RGB)
- Edges: Locations where color or brightness changes sharply
- Key points/Corners: Special points used in image stitching or matching
- Texture: Surface patterns or roughness using methods like LBP or GLCM
- Shape: Outlines and structures of objects
- Gradient and Blobs: Intensity changes and smooth regions

Feature Extraction from Images

Feature extraction transforms useful image parts into numerical representations, simplifying computational tasks like object detection, image comparison, and clustering similar images.

Pre-processing Steps:
- Import the image
- Import necessary libraries
- Convert image from BGR to RGB format
- Display the image

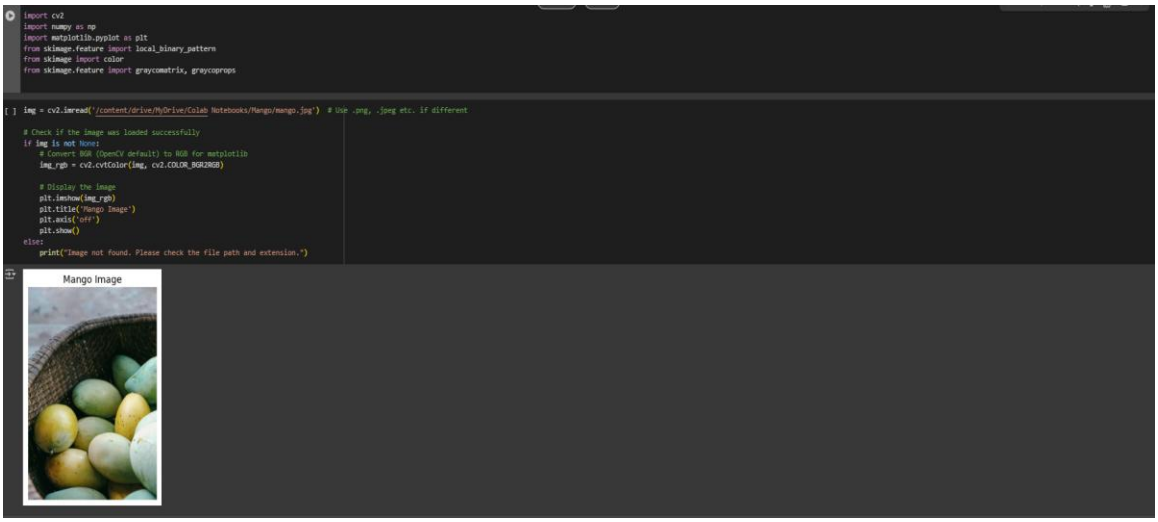

Figure 1: Mango Image

Color-Based Features - Color Histogram

A color histogram displays the frequency of different color intensities in an image, using either RGB or

HSV color spaces. Histograms are normalized to allow fair comparison between images of different sizes.

Figure 2: Color Histogram 01

Normalizing the Histograms

Normalization is important because it ensures that the histogram values are independent of the image size and total number of pixels. This allows for consistent comparison between histograms from different images, regardless of their dimensions. Normalized histograms provide a probability distribution of pixel intensities, which is useful in various image processing tasks, such as image comparison, feature extraction, and image recognition.

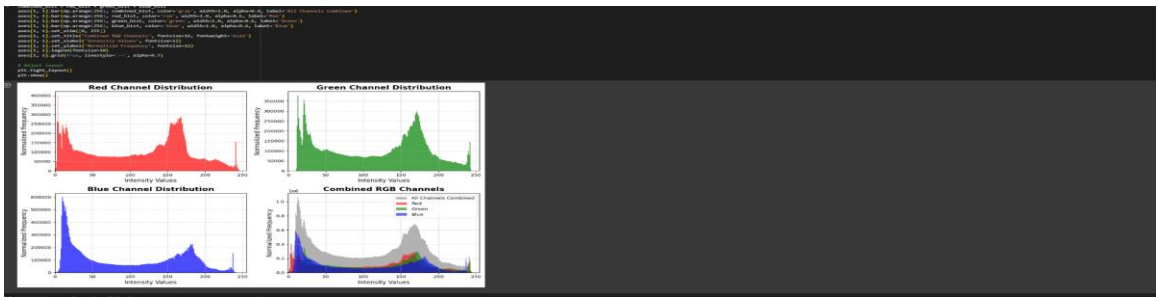

Figure 2: Color Histogram 02

GLCM (Gray-Level Co-occurrence Matrix)

GLCM measures how often pairs of pixel values occur at a certain distance and orientation.

- Contrast: Measures the intensity contrast between pixels

$$\text{Contrast} = \sum_{i,j} (i-j)^2 \cdot P(i,j)$$

where $P(i,j)$ is the (i,j)-th entry in the normalized GLCM.

- Correlation: Shows the linear dependency of gray levels

$$\text{Correlation} = \sum_{i,j} \frac{(i - \mu_i)(j - \mu_j) \cdot P(i,j)}{\sigma_i \sigma_j}$$

where μ_i and μ_j are the means, and σ_i and σ_j are the standard deviations of the GLCM.

- Energy: Measures uniformity (how often certain patterns repeat)

$$\text{Energy} = \sum_{i,j} P(i,j)^2$$

- Homogeneity: Measures of how close elements are to the GLCM diagonal (smoothness)

$$\text{Homogeneity} = \sum_{i,j} \frac{P(i,j)}{1 + |i - j|}$$

Figure 3: GLCM Texture Features Diagram

Texture-Based Features - Local Binary Patterns (LBP)

Local Binary Patterns (LBP) is a powerful and efficient method for texture classification and feature extraction in image processing. LBP encodes the local texture of an image by thresholding neighboring pixels against the center pixel. This technique effectively captures the spatial patterns or textures within an image, making it especially useful for analyzing textures and patterns in various applications such as face recognition, object detection, and image retrieval.

Steps:

1. Convert the image to grayscale
2. Compare the center pixel with its neighbors
3. Create a binary code for each pixel
4. Form a histogram of these binary codes

Figure 4: Local Binary Pattern Generation

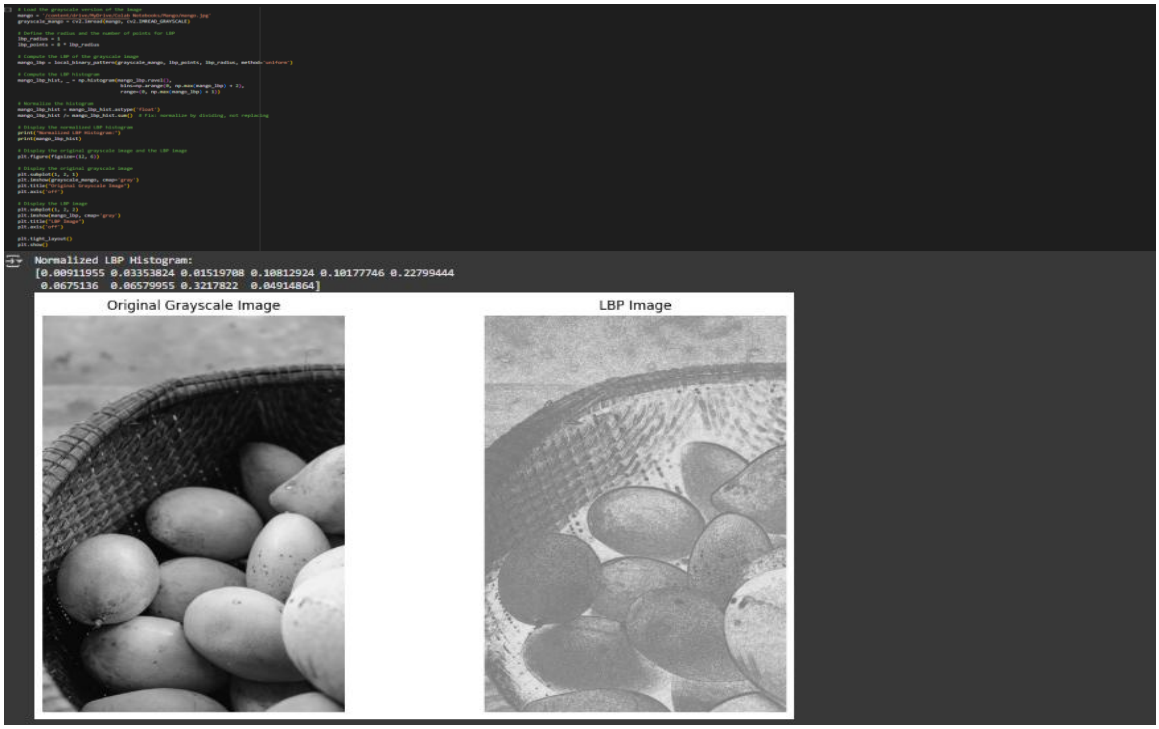

Figure 5: Illustration of Local Binary Pattern Generation

Edge Detection - Canny Edge Detection

Canny edge detection is a popular technique to detect edges in an image. It plays a critical role in applications such as object detection, image segmentation, and feature extraction. These algorithms aim to detect points where there is a sharp change in image intensity, marking the

boundaries between different regions or objects.

Steps:

1.Gaussian Smoothing

- Reduce noise in the image by applying a Gaussian filter:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Here, σ is the standard deviation, and x, yx, yx, y are the pixel coordinates.

2.Gradient Calculation

- Compute the gradients $ G_x$ and $ G_y$ using Sobel operators:

$$G_x = \frac{\partial I}{\partial x}, \quad G_y = \frac{\partial I}{\partial y}$$

- Calculate the gradient magnitude $ G$ and direction θ:

$$G = \sqrt{G_x^2 + G_y^2}, \; \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

3.Non-Maximum Suppression

- Thin the edges by retaining only the local maxima in the gradient direction:
    - For each pixel, compare its gradient magnitude to the magnitudes of its neighbors along the gradient direction. Retain the pixel only if its magnitude is the largest.

4.Double Thresholding

- Classify edges based on high (THT_HTH) and low (TLT_LTL) thresholds:
    - **Strong edges**: $G > T_H$
    - **Weak edges**: $T_L < G \leq T_H$
    - **Suppress** $G \leq T_L$

5.Edge Tracking by Hysteresis

- Link weak edges to strong edges if they are connected:

- Retain strong edges.
- Preserve weak edges only if they are connected to strong edges; otherwise, suppress them.

Figure 6: Canny Edge Detection

Corner Detection - Harris Corner Detection

Harris corner detection identifies points in an image where the intensity changes sharply in multiple directions. The Harris Corner Detection algorithm identifies corners by calculating the gradient products, applying Gaussian smoothing, computing a corner response function, and then performing thresholding and dilation to highlight the corners on the original image. The following steps outline the algorithm:

1. **Convert Image to Grayscale**
   - Convert the input image to a grayscale image:

$$I(x,y) = 0.299 \cdot R(x,y) + 0.587 \cdot G(x,y) + 0.114 \cdot B(x,y)$$

   - Where $R(x,y)$, $G(x,y)$, and $B(x,y)$ are the red, green, and blue color channels, respectively.
2. **Compute Image Gradients**
   - Calculate the gradients of the grayscale image in the x and y directions using filters (e.g., Sobel operators):

$$I_x = \frac{\partial I}{\partial x}, \quad I_y = \frac{\partial I}{\partial y}$$

3. **Compute Gradient Products**
   - Compute the products of the gradients:

$$I_x^2 = I_x \cdot I_x, \quad I_y^2 = I_y \cdot I_y, \quad I_x I_y = I_x \cdot I_y$$

4. **Apply Gaussian Filter**

- Smooth the gradient products with a Gaussian filter to obtain the components of the structure tensor:

$$M_{11} = \sum_{x,y} I_x^2 \cdot G(x, y)$$

$$M_{22} = \sum_{x,y} I_y^2 \cdot G(x, y)$$

$$M_{12} = \sum_{x,y} I_x I_y \cdot G(x, y)$$

- Where $G(x, y)$ is a Gaussian kernel with standard deviation σ.

5. **Compute Corner Response**

- Calculate the Harris corner response R:

$$R = \det(M) - k \cdot (\text{trace}(M))^2$$

  - **Determinant**:

$$\det(M) = M_{11} \cdot M_{22} - (M_{12})^2$$

  - **Trace**: $\text{trace}(M) = M_{11} + M_{22}$
  - **Constant** k is typically set to 0.04 - 0.06.

6. **Thresholding and Non-Maximum Suppression**

- Apply a threshold to the corner response to identify corners:

$$\text{Corners} = R > \text{threshold} \times R_{\max}$$

- Where $R_{\max}$ is the maximum value of R.

7. **Dilate Corners**

- Dilate the detected corners to enhance visibility: Dilated = dilate(R, None)

8. **Overlay Corners on Original Image**

- Mark the corners on the original image:
  image [corners > 0.01 * corners.max()] = [0, 0, 255]
- Corners are marked in red ([0, 0, 255] in BGR color format).

Figure 7: Harris Corner Detection Output

Other Feature Extraction Techniques

Several other methods are widely used for image feature extraction, each with unique strengths:

- CNNs (Convolutional Neural Networks): Excellent for object recognition in large-scale datasets.
- Autoencoders: Unsupervised learning models that compress and denoise image features.
- HOG (Histogram of Oriented Gradients): Effective for detecting humans and faces in low-resolution images.
- SIFT (Scale-Invariant Feature Transform): Identifies key points robust to scaling and rotation.
- Frequency-Based Methods: Fourier Transform techniques that analyze stable repetitive patterns.

Conclusion

Feature extraction forms the backbone of many computer vision applications by transforming raw image data into meaningful representations. From traditional methods like GLCM and LBP to advanced CNNs and SIFT, understanding and selecting the right feature extraction method is key to building effective image processing solutions.