American International University - Bangladesh (AIUB)

Assignment: Feature Extraction Techniques from raw image

Subject: Machine learning

Supervisor: Dr. Md. Asraf Ali

Name: ARIFUR RAHMAN

ID: 22-47900-2

Section: C

Introduction

A feature is a distinctive attribute or measurable property in data that helps describe patterns or characteristics. In machine learning, features are the inputs that represent the data and are used by models to learn relationships and make predictions. An image feature is a specific characteristic or part of an image that is relevant for understanding its content. Features help reduce the complexity of image data while retaining essential information for tasks like detection, recognition, and segmentation.

Types of Image Features

Image features are categorized based on their characteristics and the tasks they support. These categories include low-level, key point-based, global, and deep features.

1. Low-Level Features

These features capture basic image characteristics:

- Edges: Boundaries between regions with different intensities.
- Corners: Intersection points of edges, essential for pattern recognition.
- Texture: Repeating patterns or surface properties in an image.

2. Key points and Descriptors

These features focus on distinctive points within an image:

- Key points: Unique and stable points in an image, such as corners or blobs
- Descriptors: Mathematical representations of key points, used for comparing and matching key points across images.

3. Global Features

These features describe the entire image as a whole:

- Histograms: Representations of pixel intensity distributions.
- Fourier Descriptors: Frequency-based features useful for shape analysis.
- Moments: Statistical properties like Hu Moments, used to describe image shape and orientation.

4. Deep Features

These are high-level features learned automatically through deep learning models:

- Convolutional Neural Networks (CNNs): Deep learning models that learn hierarchical features, capturing complex patterns like shapes, textures, and objects, typically used for image classification, object detection, and segmentation.

These feature types serve various computer vision tasks, from simple image processing to advanced object recognition.

Feature Extraction Techniques

1. Color Histogram

A color histogram represents the color distribution within an image by counting the pixel intensities across different color channels. In this study, we computed the histogram using cv2.calcHist with 8 bins for each of the BGR channels, normalizing the histogram to produce a feature vector.

- **Purpose**: To represent the color distribution within an image.
- **Method**: Generates histograms for pixel intensities across the BGR color channels.
- **Output**: A normalized feature vector representing color distribution.

Code:

```
for channel, color in zip(channels, colors):
    hist = cv2.calcHist([channel], [0], None, [256], [0, 256])
    plt.plot(hist, color=color)
    plt.xlim([0, 256])
```

2. **Edge Detection (Canny)**

Edge detection identifies the boundaries in an image by highlighting regions of rapid intensity change. We used the Canny edge detector applied to the grayscale image to extract these edges.

- **Purpose:** To detect image boundaries by identifying areas with rapid intensity changes.
- **Method**: Applies the Canny edge detector to a grayscale image.
- **Output**: An edge-detected image that highlights the boundaries.

Code:

```
# 2. Edge Detection (Canny)
edges = cv2.Canny(gray_image, 100, 200)
```

3. **Local Binary Pattern (LBP)**

LBP is a texture descriptor that captures the local structure of an image by comparing each pixel with its neighboring pixels, generating a binary pattern. The histogram of these patterns forms a feature vector representing the texture.

- **Purpose**: To capture texture information by comparing pixel intensities with neighbors. **Method**: Computes a binary pattern for each pixel and generates a histogram of these patterns. **Output**: An LBP histogram representing the texture.

Code:

```
# 3. Texture Features (Local Binary Pattern)
lbp = feature.local_binary_pattern(gray_image, P=8, R=1, method='uniform')
n_bins = int(lbp.max() + 1)
lbp_histogram, _ = np.histogram(lbp, density=True, bins=n_bins, range=(0, n_bins))
```

4. **Contour Detection**

Contours are curves that connect continuous points along a boundary with the same color or intensity. We used cv2.findContours to detect contours from the edge-detected image and then drew these contours on the original image.

- **Purpose**: To find and highlight object boundaries within an image.
- **Method**: Detects contours in the edge-detected image and overlays them on the original image. **Output**: An image with contours drawn around the objects.

Code:

```
# 4. Shape Features (Contour Detection)
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contour_image = image.copy()
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)
```

5. Keypoint Detection (ORB)

ORB (Oriented FAST and Rotated BRIEF) is a robust technique for feature detection and description. It identifies key points in the image and computes their descriptors, making it useful for tasks such as image matching and object recognition.

- **Purpose**: To identify and describe key points for image matching and recognition tasks.
- **Method**: Uses ORB to detect key points and compute descriptors. **Output**: Key points and descriptors for further analysis or matching.

Code:

```
# 5. Keypoint Detection (ORB)
orb = cv2.ORB_create()
keypoints, descriptors = orb.detectAndCompute(gray_image, None)
```

6. Deep Learning Features (ResNet)

We employed a pre-trained ResNet-50 model (excluding the final classification layer) to extract deep features from the image. This model, trained on a large dataset, provides high-level features that capture complex patterns and structures.

- **Purpose**: To extract high-level features that capture complex patterns and structures.
- **Method**: Utilizes a pre-trained ResNet-50 model to extract features after image preprocessing. **Output**: A feature vector representing the deep features learned by the model.

Code:

```
# 6. Deep Learning Features (ResNet)
model = models.resnet50(pretrained=True)
model = torch.nn.Sequential(*list(model.children())[:-1])
model.eval()

preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Application and Use Cases:

1 - Color Histogram and Deep Learning Features are particularly useful for image classification tasks, where the goal is to categorize images based on their content.

2 - Edge Detection, LBP, and Contour Detection are valuable for identifying object boundaries, textures, and shapes within an image, which is crucial for object detection and segmentation tasks.

3 – Key point Detection is especially important in feature matching applications, such as aligning images or recognizing objects across different scenes.

Table:

Feature Extraction Technique	Steps Involved	Output
Color Histogram	1. Original Image (RGB)	Feature Vector (Normalized)
	2. Color Histogram	
	Computation	
Edge Detection (Canny)	1. Original Image (RGB)	Edge Image
	2. Convert to Grayscale	
	3. Canny Edge Detection	
Local Binary Pattern (LBP)	1. Original Image (RGB)	LBP Histogram
	2. Convert to Grayscale	
	3. LBP Computation	
	4. LBP Image	
Contour Detection	1. Original Image (RGB)	Contour Image

	2. Convert to Grayscale	
	3. Edge Detection	
	4. Find Contours	
Keypoint Detection (ORB)	1. Original Image (RGB)	Keypoints & Descriptors
	2. Convert to Grayscale	
	3. ORB Detector	
Deep Learning Features (ResNet)	1. Original Image (RGB)	Feature Vector
	2. Preprocessing	
	3. Pre-trained ResNet Model	

Result:

Original Image:

Fig 01: Original Image.

Color Histogram:

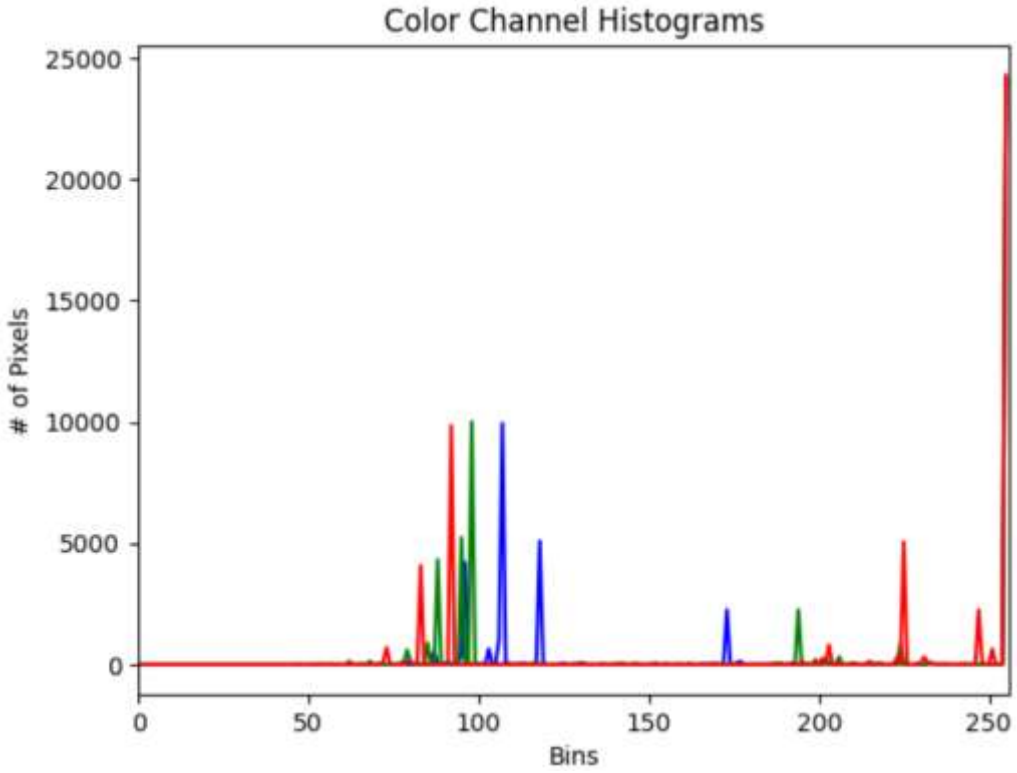

Fig 02: Color Histogram.

Edge Detection (Canny):

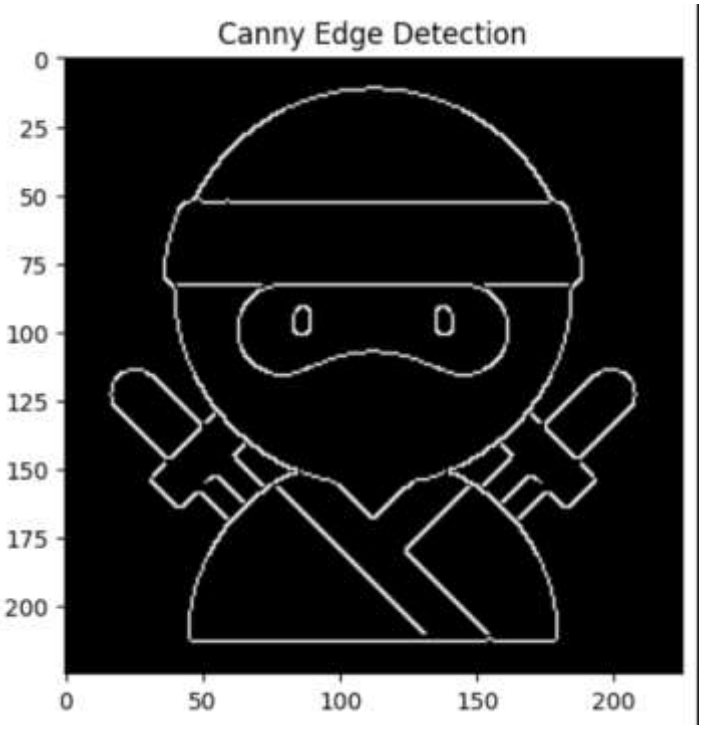

Fig 03: Edge Detection (Canny)

Local Binary Pattern (LBP):

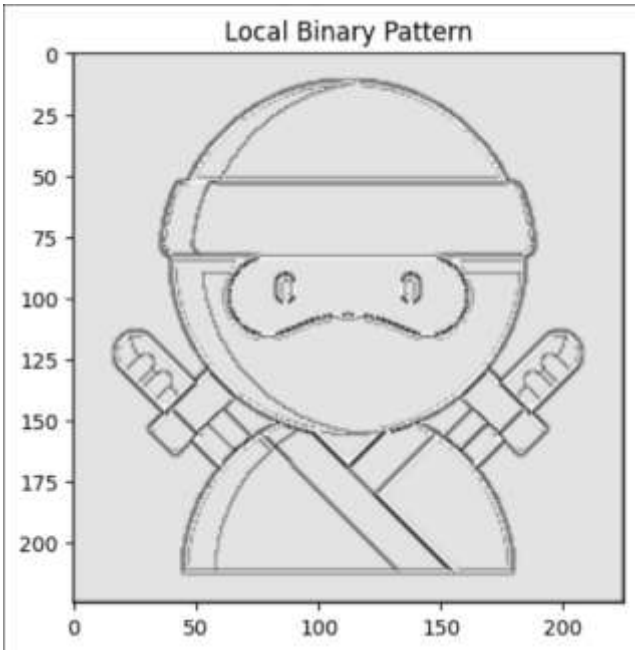

Fig 04: Edge Detection (Canny)

Contour Detection:

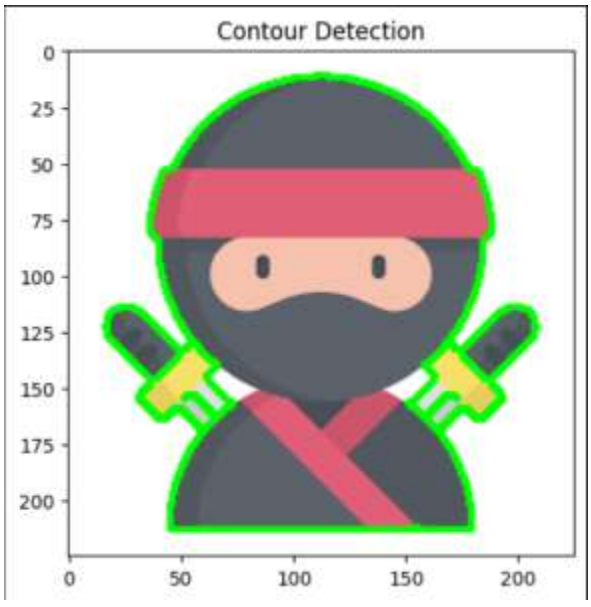

Fig 05: Edge Detection (Canny)

Keypoint Detection (ORB):

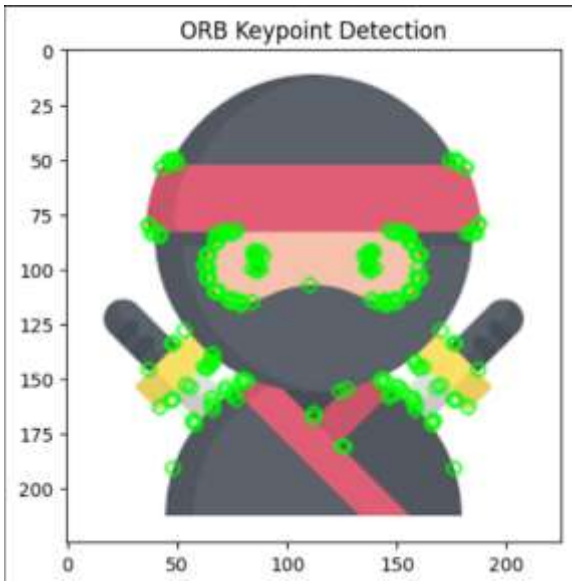

Fig 06: Edge Detection (Canny)

Deep Learning Features (ResNet):

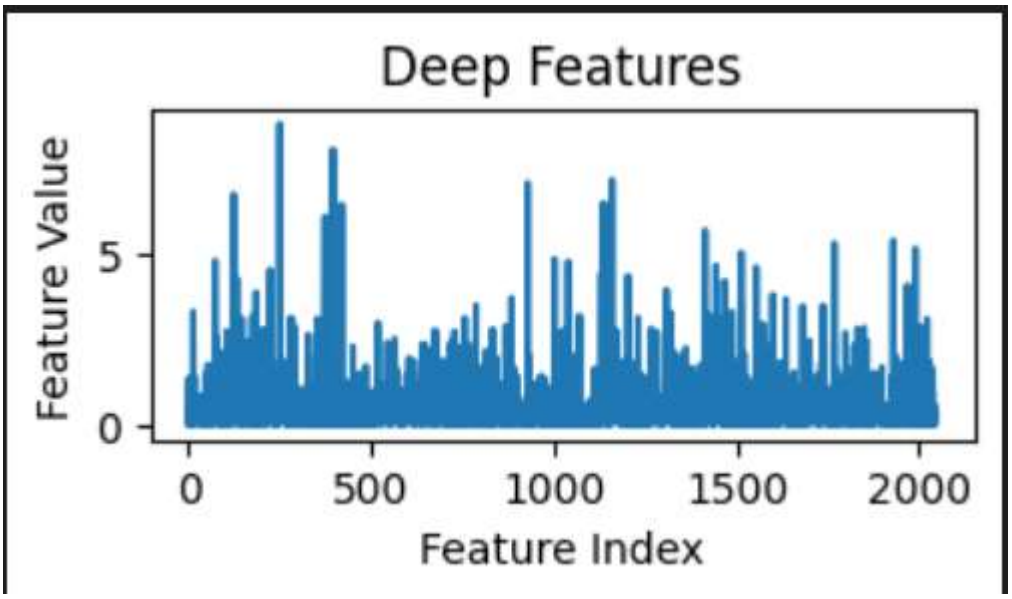

Fig 07: Edge Detection (Canny)

Discussion

The effectiveness of the feature extraction techniques varies depending on the image analysis task. The color histogram captures color distribution efficiently, making it suitable for color-based image classification. The Canny edge detector effectively emphasizes object boundaries, which is advantageous for segmentation and shape analysis. Local Binary Pattern (LBP) represents texture by comparing pixel intensities, though it can be challenged by noise and complex textures. Contour detection, derived from Canny edges, outlines object shapes clearly, aiding object recognition. ORB key point detection is robust for feature matching and object recognition, adapting well to rotation and scale changes. Lastly, deep learning features from ResNet-50 provide high-level, complex image representations, making them ideal for advanced classification and detection. Each technique has its own strengths and limitations, and their use depends on the specific requirements of the computer vision task.

Conclusion

Each feature extraction technique possesses unique strengths and limitations, making them appropriate for different computer vision tasks. While color histograms and deep learning features excel in image classification, edge detection, LBP, and contour detection are essential for object detection and texture analysis. Key point detection plays a critical role in matching and recognition tasks. Selecting the appropriate technique depends on the specific application and the characteristics of the image data being analyzed.