## **Problem Description**

The goal is to implement two key functionalities:

1. Animal Detection: Detect animals in an image using the YOLOv8 model.
2. Face Profiling: Identify facial landmarks in an image using Dlib.

## **Allowed Operations**

* Animal Detection
  + Load a pre-trained YOLOv8 model.
  + Process an image and detect animals.
  + Draw bounding boxes with labels around detected animals.
  + Display the processed image.
* Face Profiling
  + Detect faces using Dlib's face detector.
  + Identify 68 facial landmarks.
  + Mark each landmark on the face.
  + Display the processed image.

## **Approach**

### **1. Animal Detection with YOLOv8**

* Load the YOLOv8 model (yolov8n.pt).
* Read the input image using OpenCV (cv2.imread).
* Use the YOLO model to predict objects in the image.
* Extract bounding box coordinates, class labels, and confidence scores.
* Draw bounding boxes and labels for detected animals with confidence above 0.5.
* Display the processed image using OpenCV (cv2.imshow).

### **2. Face Profiling with Dlib**

* Load the Dlib frontal face detector.
* Use a pre-trained facial landmark predictor (shape\_predictor\_68\_face\_landmarks.dat).
* Convert the image to grayscale and detect faces.
* For each detected face, extract 68 facial landmarks.
* Mark landmarks as red dots using OpenCV (cv2.circle).
* Display the image with marked landmarks.

## **Explanation**

### **1. Animal Detection**

|  |
| --- |
| model = YOLO("yolov8n.pt") # Load YOLOv8 model |

* Loads the pre-trained YOLOv8 model for object detection.

|  |
| --- |
| img = cv2.imread(image\_path) # Read the image  results = model(img) # Run YOLO detection on the image |

* Reads the input image and passes it to the YOLO model for inference.

|  |
| --- |
| for result in results:  for box in result.boxes:  x1, y1, x2, y2 = map(int, box.xyxy[0]) # Get bounding box coordinates  label = result.names[int(box.cls[0])] # Get class label  conf = box.conf[0] # Get confidence score  if conf > 0.5: # Filter detections with confidence > 0.5  cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2) # Draw bounding box  cv2.putText(img, label, (x1, y1 - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2) # Label the detection |

* Loops through detected objects and extracts:
  + Bounding box coordinates
  + Class label
  + Confidence score
* Draws a bounding box and label if confidence > 0.5.

### **2. Face Profiling**

detector = dlib.get\_frontal\_face\_detector() # Load face detector

predictor = dlib.shape\_predictor("shape\_predictor\_68\_face\_landmarks.dat") # Load facial landmark predictor

* Initializes Dlib's face detector and landmark predictor.

|  |
| --- |
| img = cv2.imread(image\_path) # Read the image  gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) # Convert to grayscale  faces = detector(gray) # Detect faces |

* Converts the image to grayscale for better face detection.
* Detects faces using the Dlib face detector.

|  |
| --- |
| for face in faces:  landmarks = predictor(gray, face) # Get facial landmarks  for n in range(68): # Loop through 68 landmark points  x, y = landmarks.part(n).x, landmarks.part(n).y # Get landmark coordinates  cv2.circle(img, (x, y), 2, (0, 0, 255), -1) # Draw red dots on landmarks |

* Extracts 68 facial landmarks and marks each point on the face.

## **Expected Output**

1. Animal Detection Output
   * An image with detected animals highlighted by green bounding boxes and labeled.
2. Face Profiling Output
   * An image with 68 facial landmarks marked as red dots.

### **Example Outputs**

* Animal Detection:  
  🟩 Bounding boxes around detected animals with labels (e.g., "dog", "cat").
* Face Profiling:  
  🔴 68 red dots on detected facial landmarks.