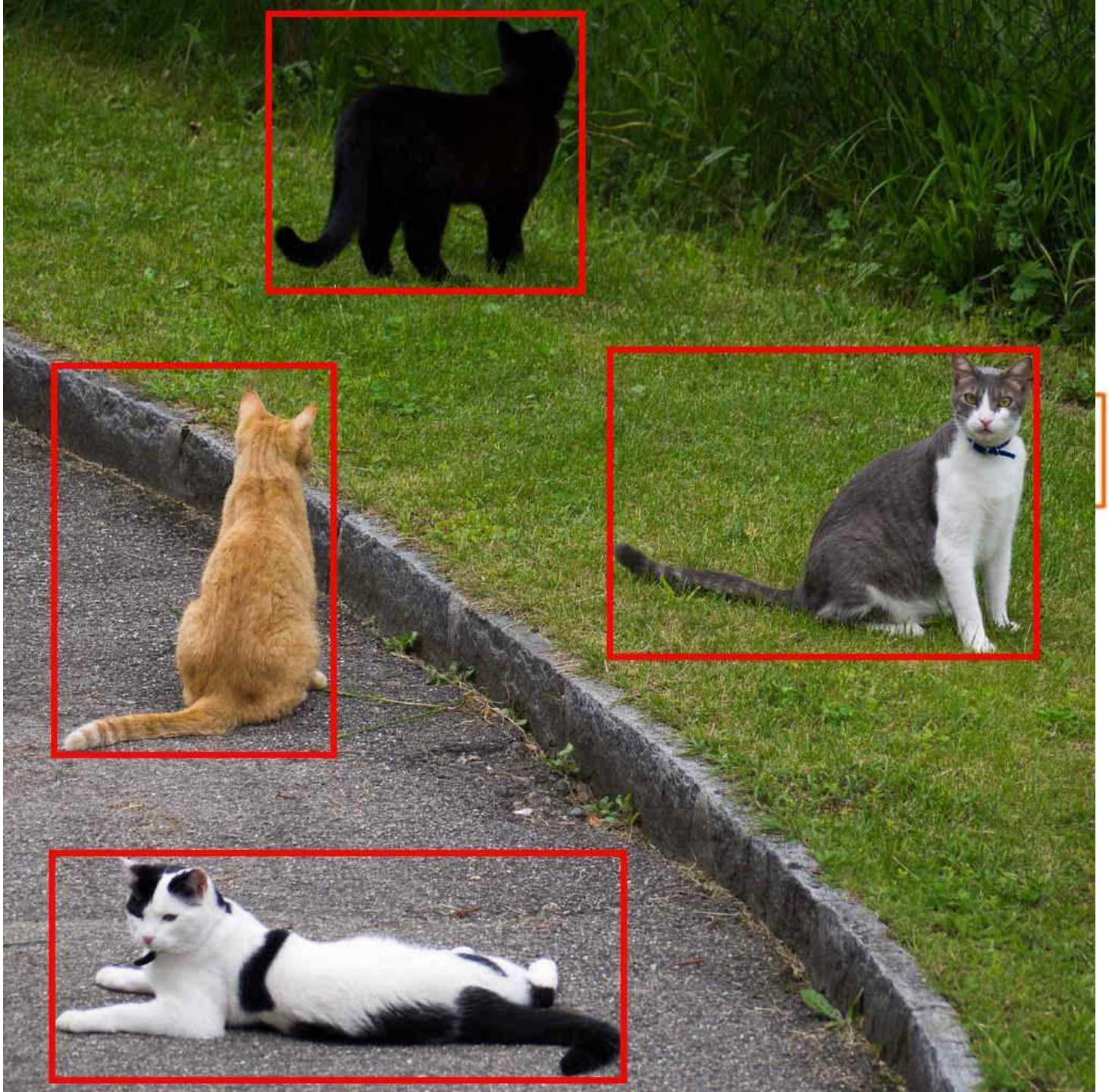
Data Collection with OpenCV & Python

Gesture Recognition using ROI
(Region of Interest)

CS 428

Learning Objectives

- - Understand how to collect gesture data using Python and OpenCV.
- - Configure video streams and handle external cameras.
- - Save Regions of Interest (ROI) from video frames.
- - Prepare the collected data for further analysis or training.



Introduction and Motivation

- In modern AI applications, gesture recognition is crucial in enabling intuitive user interfaces.
- This project demonstrates how we can collect gesture data to build a machine learning model for gesture classification.
- By using OpenCV and Python, we will learn to automate the data collection process from video streams.
- This will form the foundation for building real-world AI solutions, such as controlling devices using hand gestures.

Step 1 - Argument Parsing with argparse

- We use argparse to manage command-line inputs efficiently.
- The script accepts a --conf argument, which points to the JSON configuration file.
- This configuration file contains key details about the video source, gesture mappings, and dataset paths.
- Example Command: python gather_examples.py --conf config/config.json

Step 2 - Loading the Configuration File

1. The configuration file, loaded using the Conf class, contains critical parameters:

   - Gesture capture area coordinates.
   - Mappings between keys and gestures.
   - Dataset storage paths for saving gesture images.

2. This simplifies code management by isolating configurable parameters in one place.

Step 3 - Defining the Gesture Capture Area

- The region where the hand gesture will be detected is defined by top-left and bottom-right coordinates. These coordinates come from the configuration file. This defines a rectangular area of interest in the video frame for gesture recognition.
  The hand gestures within this area will be captured and processed.
- Example Code: TOP_LEFT = tuple(conf['top_left']) BOT_RIGHT = tuple(conf['bot_right'])

Step 4 - Gesture Label Mappings

- The configuration file provides key-to-gesture mappings.
  These mappings are converted into a form compatible with the operating system.
  This allows the code to register key presses and associate them with specific gestures.
- Example Code: MAPPINGS[ord(key)] = label
  validKeys = set(MAPPINGS.keys())

Step 5 - Video Stream Initialization

- Using the `imutils` and OpenCV libraries, we initialize the video stream.
  The script checks for the presence of an external camera, such as **OrbiCam**.
  If not detected, it defaults to the PiCamera.
  This enables compatibility across different hardware setups.
- Example Code: vs = VideoStream(src=0).start()

Step 6 - Region of Interest (ROI) Processing

- After reading a frame from the video stream, we resize and flip it for better usability.
  We then extract the Region of Interest (ROI), where the hand gesture is located.
- The ROI is converted to grayscale and thresholded to highlight the hand area, which is crucial for gesture recognition.
- Example Code: roi = frame[TOP_LEFT[1]:BOT_RIGHT[1], TOP_LEFT[0]:BOT_RIGHT[0]]
  roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
  roi = cv2.threshold(roi, 75, 255, cv2.THRESH_BINARY)[1]

Step 7 - Capturing Gesture Data

- Once a key corresponding to a gesture is pressed, the script saves the ROI as an image. This image is saved in a folder corresponding to the gesture label.
  The folder structure helps in organizing data for training gesture recognition models.
- Example Code: cv2.imwrite(p, roi)

Step 8 - Storing Collected Data

- Each gesture image is stored in a folder named after the gesture label.
  The images are stored with incremental file names to ensure no overwriting.
  This data will later be used for training a gesture recognition model.
- Example Code: p = os.path.sep.join([conf['dataset_path'], MAPPINGS[key]])

Summary of Key Steps

- 1. Parse configuration for video and gesture mappings.
  2. Define ROI for gesture capture.
  3. Capture gesture images and store them in labeled directories.
  4. These steps help create a labeled dataset for gesture recognition.

Next Steps

- - Collect enough gesture data for each label.
  - Ensure the data is clean and well-organized.
  - Use the dataset to train a machine learning model for gesture recognition.