DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF AGRICULTURE

FAISALABAD

Visual Programming

Object Detection Project

Submitted By:

Muhammad Imran (2021-ag-7808)

Muhammad Ilyas Chishti (2021-ag-7813)

Sohaib Khawar (2021-ag-7832)

SUBMITTED TO:

SIR. AHSAN RAZA SATTAR

Table of Contents

Introduction of group members:

Muhammad Imran:

I am a python developer. My role in this project is to describe below.

- to implement the code that performs object detection in an image.
- Researching and understanding object detection algorithms YOLO, or mobile-net SSD.
- Implementing the chosen object detection algorithm using Python and OpenCV library
- Loading the trained object detection model and applying it to the input image to detect objects.
- Extracting relevant information about the detected objects, such as their labels, bounding box coordinates, and confidence scores.

Muhammad Ilyas Chishti:

Interface Creation:

- Researching and selecting an appropriate framework or library for UI/GUI development in Python, such as Tkinter.
- Designing an intuitive and user-friendly interface that allows users to input an image or video and view the detected objects.
- Implementing the chosen UI framework to create the interface, including buttons, image display areas, and output panels.

Sohaib Khawer:

- My role in this project is to write documentation and user manual.
- Describing the overall architecture of the object detection system, including its various components and how they interact.
- Providing step-by-step instructions on how to set up and run the project, including the installation of dependencies, and required libraries.

Introduction of Project:

The "Object Detection" project is a Python-based application that utilizes various libraries such as OpenCV, Tkinter, and NumPy. It aims to provide an efficient and user-friendly system for object detection tasks. This documentation will provide a comprehensive overview of the project, including its objectives, the development environment, a list of modules used, and the benefits offered by the new system.

Project Objectives:

- Develop a robust object detection system using Python.
- Implement computer vision techniques to detect and recognize objects in images or video streams.
- Enhance accuracy and efficiency in object detection compared to previous systems.
- Provide a user-friendly interface for easy interaction and control.
- Enable real-time object detection and tracking for various applications.

Development Environment:

- The project is developed using Python programming language and relies on several external libraries:
- OpenCV: A powerful computer vision library that provides various tools and algorithms for image and video processing, including object detection.
- Tkinter: A standard Python GUI toolkit used to create the graphical user interface (GUI) of the application.
- NumPy: A fundamental library for scientific computing in Python, used for efficient numerical operations and array manipulation.

List of Modules:

- OpenCV: Used for object detection, image processing, and computer vision tasks.
- Tkinter: Utilized for creating graphical user interface (GUI) elements and interaction.

- NumPy: Employed for efficient numerical computations and manipulation of arrays and matrices.

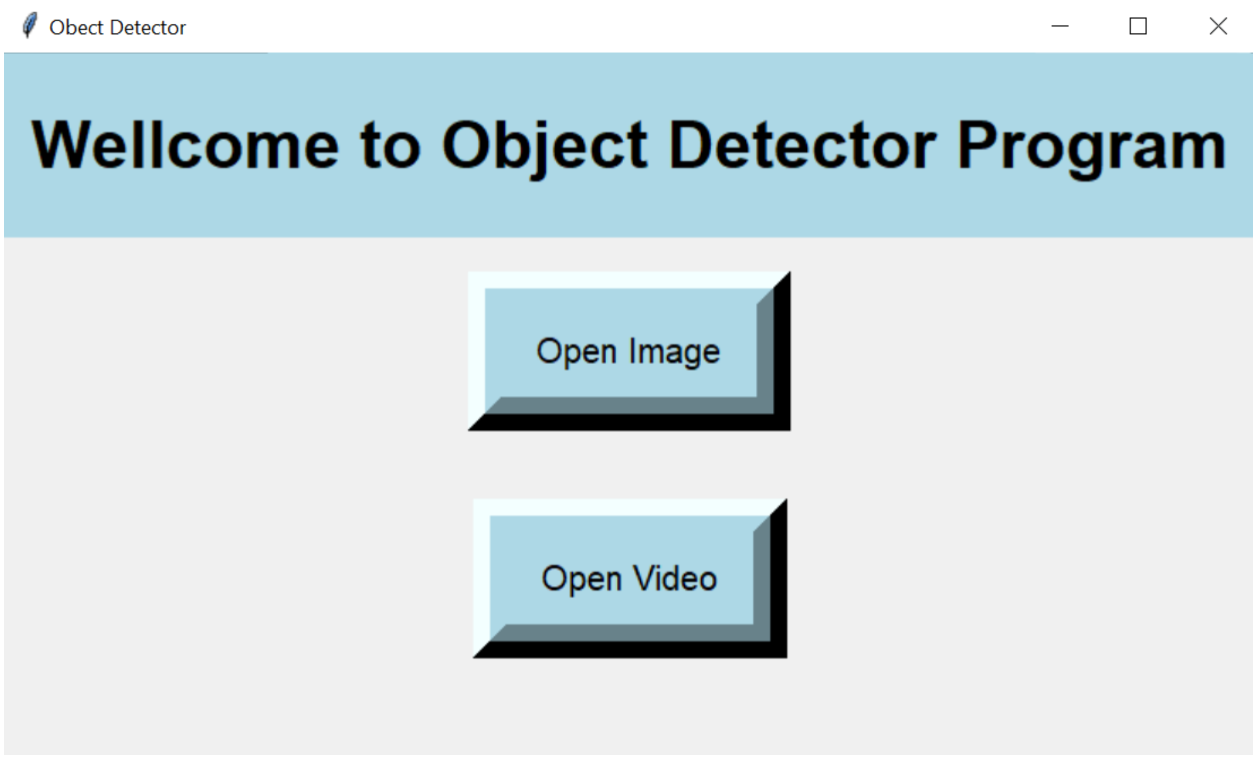
Benefits of the New System:

- The new "Object Detection" system offers several advantages over the previous system and other alternative approaches:
- Improved accuracy: By leveraging computer vision techniques and the capabilities of OpenCV, the system enhances the accuracy of object detection.
- Real-time detection: The system is designed to perform object detection and tracking in real-time, making it suitable for time-critical applications.
- User-friendly interface: The integration of Tkinter allows for a user-friendly GUI, enabling easy interaction and control of the system.
- Customizability: The modular nature of the project allows for easy integration of additional functionalities and customization as per specific requirements.
- Efficiency: By utilizing NumPy and optimizing the code, the system achieves efficient processing and execution.

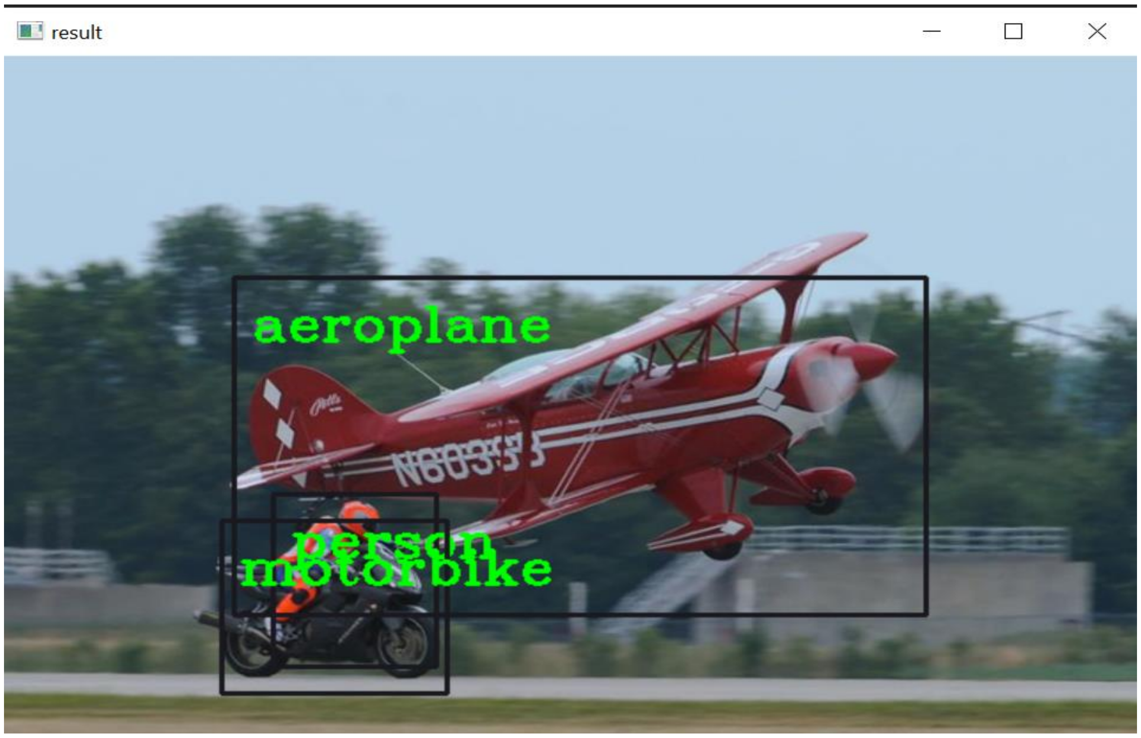
This documentation provides a comprehensive overview of the "Object Detection" project, including its objectives, development environment, modules used, and the benefits it offers. It serves as a valuable resource for understanding the project and its functionalities.

Results and Discussion:

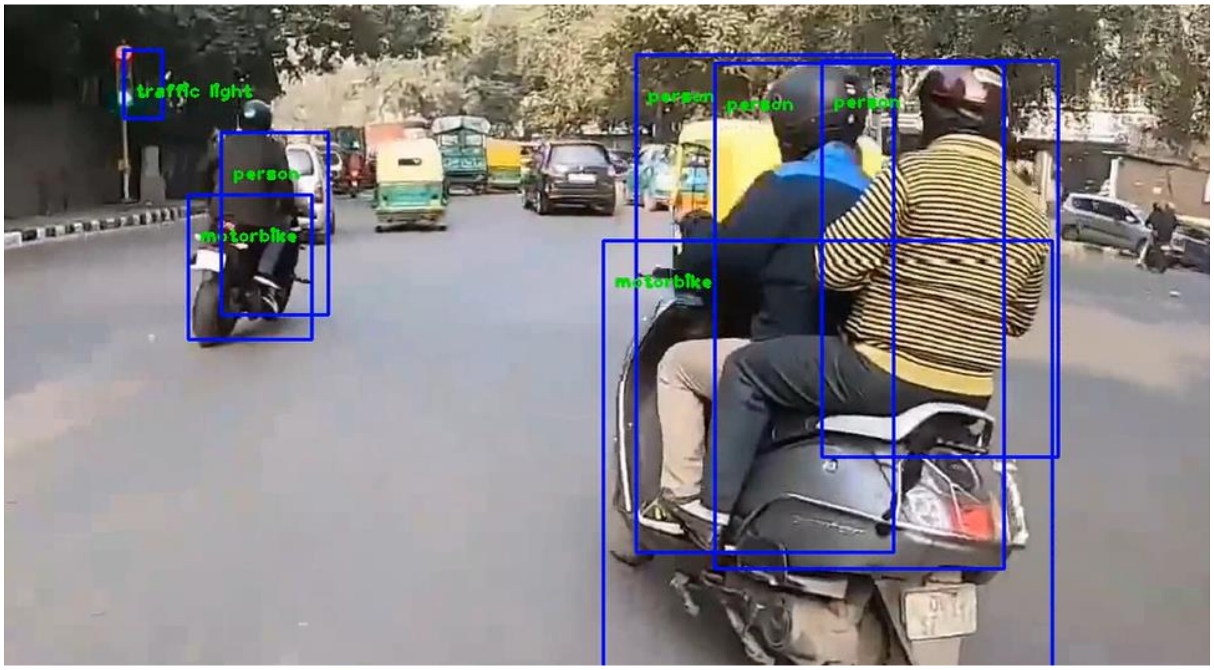
This is the basic interface of the object detector application:



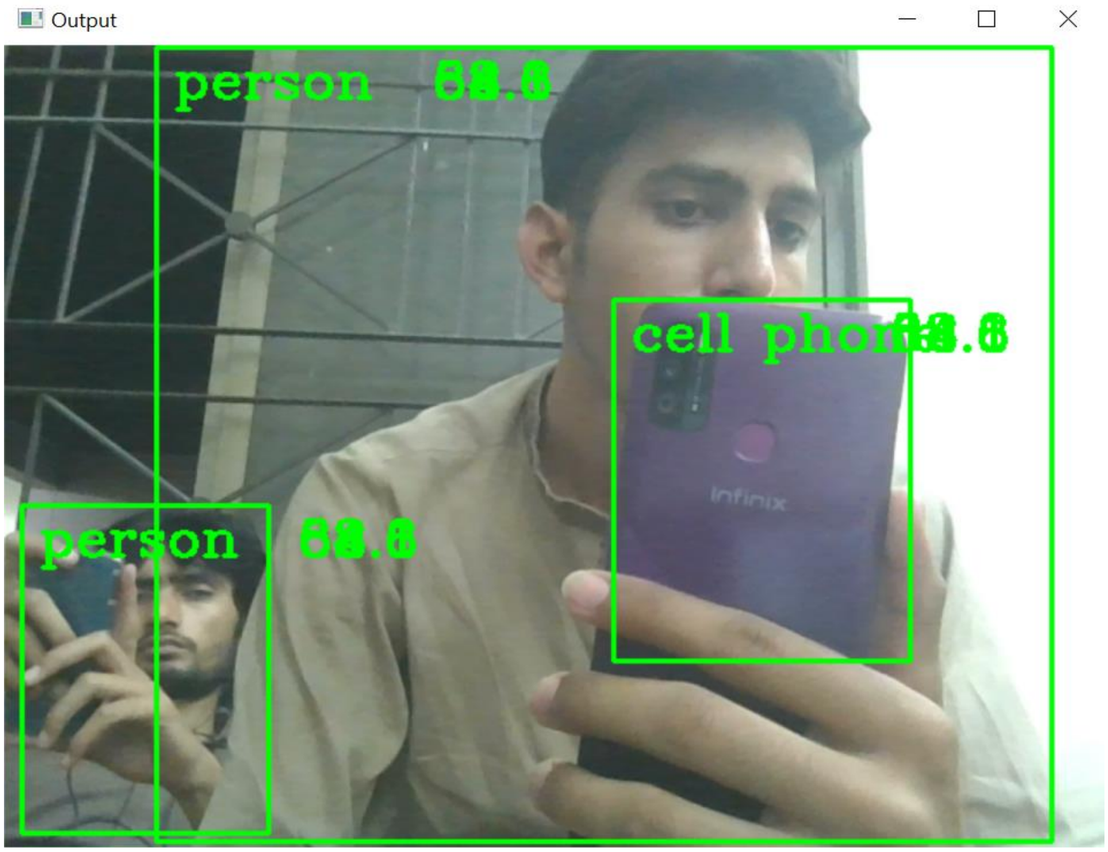
Object detection in picture



Object detection in Video:



Object Detection using webcam.



User Manual:

1. Introduction:

The Object Detection program is a user-friendly application developed in Python, utilizing libraries such as OpenCV, Tkinter, and NumPy. This user manual will guide you through the process of running the program, even if you have no prior knowledge of programming.

2. System Requirements:

To run the Object Detection program, ensure that you have the following requirements:

- Operating System: Windows, macOS, or Linux
- Python installed (version 3.x)
- Required Python libraries: cv2, tkinter, numpy

3. Installation:

Follow these steps to set up the program:

a. Install Python: Download and install the latest version of Python from the official Python website (https://www.python.org).

b. Install required libraries: Open a command prompt or terminal and run the following commands to install the necessary libraries:

```
pip install opencv-python
pip install numpy
```

4. Running the Program:

a. Launch the program: Open a command prompt or terminal and navigate to the directory where the program code is saved.

b. Execute the program: Run the following command to start the Object Detection program:

```
python object_detection_program.py
```

5. Program Interface:

Upon launching the program, a graphical user interface (GUI) will appear with the following elements:

- "Open Image" button: Click this button to select an image file for object detection.
- "Open Video" button: Click this button to select a video file for object detection.

6. Object Detection:

a. Image Detection:

- Click the "Open Image" button. A file dialog will open.
- Browse and select an image file (JPEG, JPG, or PNG format) you want to analyze.
- The program will load the object detection model and process the selected image.
- A new window will open, displaying the image with bounding boxes around detected objects and labels.
- Press any key to close the result window.
- The resulting image with bounding boxes will be saved as "result.png" in the same directory as the program.

b. Video Detection:

- Click the "Open Video" button. A file dialog will open.
- Browse and select a video file (MP4, AVI, or MOV format) you want to analyze.
- The program will load the object detection model and start processing the video.
- A new window will open, showing the processed video with bounding boxes around detected objects and labels.
- The program will save the resulting video as "Video_Output.avi" in the same directory as the program.
- To stop the video processing, close the result window.

Note: Ensure that the object detection model files ("ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt", "frozen_inference_graph.pb") and the class labels file ("labels.txt") are present in the same directory as the program.

7. Troubleshooting:

- If you encounter any issues running the program or selecting files, ensure that you meet the system requirements and have followed the installation steps correctly.

- Make sure the required model files and class labels file are present in the program directory.
- If the program crashes or freezes, close the program and try running it again.

Congratulations! You have successfully installed and run the Object Detection program. Enjoy detecting objects in images and videos with ease!

Complete Code:

```
import cv2
import tkinter as tk
from tkinter import filedialog


def open_image_dialog():

    file_path = filedialog.askopenfilename(filetypes=[("Image files",
"*.jpg;*.jpeg;*.png")])
    if file_path:
        config_file = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
        frozen_model = 'frozen_inference_graph.pb'

        model = cv2.dnn_DetectionModel(frozen_model, config_file)

        classLabels = []
        filename = 'labels.txt'
        with open(filename, 'rt') as spt:
            classLabels = spt.read().rstrip('\n').split('\n')

        # greater this value better the reults tune it for best output
        model.setInputSize(440, 450)
        model.setInputScale(1.0/127.5)
        model.setInputMean((127.5, 127.5, 127.5))
        model.setInputSwapRB(True)

        img = cv2.imread(file_path)

        classIndex, confidence, bbox = model.detect(
            img, confThreshold=0.5)  # tune confThreshold for best results

        font = cv2.FONT_HERSHEY_COMPLEX

        for classInd, conf, boxes in zip(classIndex.flatten(),
confidence.flatten(), bbox):
```

```
            cv2.rectangle(img, boxes, (30, 25, 25), 2)
            cv2.putText(img, classLabels[classInd-1], (boxes[0] + 10, boxes[1] +
40),
                        font, fontScale=1, color=(0, 255, 0), thickness=2)

        cv2.imshow('result', img)
        cv2.waitKey(0)

        cv2.imwrite('result.png', img)


def open_video_dialog():
    file_path = filedialog.askopenfilename(
        filetypes=[("Video files", "*.mp4;*.avi;*.mov")])
    if file_path:
        config_file = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
        frozen_model = 'frozen_inference_graph.pb'

    model = cv2.dnn_DetectionModel(frozen_model, config_file)

    classLabels = []
    filename = 'labels.txt'
    with open(filename, 'rt') as spt:
        classLabels = spt.read().rstrip('\n').split('\n')

    # greater this value better the reults but slower. Tune it for best results
    model.setInputSize(320, 320)
    model.setInputScale(1.0/127.5)
    model.setInputMean((127.5, 127.5, 127.5))
    model.setInputSwapRB(True)

    cap = cv2.VideoCapture("test_video.mp4")
    ret, frame = cap.read()

    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    # 25 is the frame rate of output video you can change it as required
    video = cv2.VideoWriter('Video_Output.avi', fourcc, 24,
                            (frame.shape[1], frame.shape[0]))

    font = cv2.FONT_HERSHEY_PLAIN

    try:
        while (True):

            ret, frame = cap.read()
```

```python
            classIndex, confidence, bbox = model.detect(
                frame, confThreshold=0.60)  # tune the confidence  as required
            if (len(classIndex) != 0):
                for classInd, boxes in zip(classIndex.flatten(), bbox):
                    if (classInd <= 80):
                        cv2.rectangle(frame, boxes, (255, 0, 0), 2)
                        cv2.putText(frame, classLabels[classInd-1], (boxes[0] +
10,
                                    boxes[1] + 40), font, fontScale=1, color=(0,
255, 0), thickness=2)

            video.write(frame)
            cv2.imshow('result', frame)
            cv2.waitKey(2)

    except:
        cap.release()
        video.release()
        cv2.destroyAllWindows()


root = tk.Tk()

root.geometry("1360x768")
root.title("Obect Detector")

wellcome_text = tk.Label(text="Wellcome to Object Detector
Program",background="lightblue", padx=15,pady=30,font=("comicsansms",29,"bold"))
wellcome_text.pack()

button_style = {"background": "lightblue", "foreground": "black", "padx": 15,
"pady": 10, "font":"comicsansms","border":20}
image_button = tk.Button(root, text="Open Image", command=open_image_dialog,
**button_style)
image_button.pack(pady=20)

video_button = tk.Button(root, text="Open Video", command=open_video_dialog,
**button_style)
video_button.pack(pady=20)

root.mainloop()
```