Real-time Room Occupancy Monitoring and Counting System with Limit Enforcement: Object Detection with OpenCV and Yolov5

Bagasbas, Nethan Gabriel B.

Sedoriosa, Febron Jr. B.

In Partial Fulfillment of the

Requirements for the CS121

Computer Programming 2

Department of Computer Science
College of Information Technology and Computing
University of Science and Technology of Southern Philippines
Cagayan de Oro City, 9000

ACKNOWLEDGEMENT

We would like to convey our sincere gratitude to all the people and organizations who helped this research project on Room Occupancy Monitoring: Object Detection with OpenCV and Yolov5 come to fruition.

First and foremost, we would like to express our gratitude to our parents for their enormous support, love, and encouragement during our academic endeavors. Their faith in us has served as a constant inspiration.

We are incredibly appreciative of our partner's commitment, teamwork, and joint efforts on this project. Our collaboration has been essential for overcoming obstacles, sparking innovation, and producing significant outcomes. We have advanced our collective knowledge and expertise.

We would like to thank ChatGPT, the outstanding language model created by OpenAI, for its priceless assistance and extensive knowledge base. We have gained insightful knowledge and information from ChatGPT, which has improved our comprehension of the topic.

Special mention should be made of the important debates, comments, and discoveries made by our peers and colleagues throughout this project process. Their suggestions have extended our horizons and improved the caliber of our work as a whole.

The OpenCV and Yolov5 communities now in existence deserve our appreciation. Their ongoing work to advance computer vision technology has provided the framework for our investigation. Our implementation process has been greatly facilitated by their resources, documentation, and open-source contributions, for which we are grateful.

Finally, we would like to express our profound gratitude to all the contributors who voluntarily gave their time and assistance to this study. Their participation has been essential in gathering the information required and confirming the efficacy of our room occupancy monitoring system.

We owe a sincere debt of gratitude to everyone who has helped us, whether directly or indirectly. Your input has been really helpful in forming this study project. We appreciate your participation in our journey.

May 2023

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

1.1 Background of the Study

The COVID-19 epidemic has significantly changed how people live, work, and interact with one another in communities all across the world. Due to the virus's high contagiousness, strict safety procedures had to be put in place to stop it from spreading. In order to maintain public health and safety, governments, organizations, and people all had to adjust to new rules and regulations.

Monitoring room occupancy or count is a crucial part of preventing the spread of the infection. It is essential to keep track of the number of people in a particular space in a variety of situations, including offices, schools, restaurants, and public places, in order to avoid crowding and uphold physical distance norms. This observation enables efficient crowd control, lowers the chance of viral spread, and aids in the overall containment of the infection.

The significance of safety procedures

To stop the spread of COVID-19, safety procedures must be put in place, including monitoring room occupancy. Organizations and people can help the overall effort to protect the public's health by following these protocols.

Reducing the risk of transmission: COVID-19 mainly spreads through close contact with infected people. Organizations can enforce occupancy limitations and make sure that physical distance rules are observed by keeping track of room occupancy. This lessens the possibility of viral transmission and aids in preserving both the health and welfare of custom

It is impossible to stress the importance of safety procedures, such as room occupancy monitoring, as the COVID-19 epidemic continues to spread throughout the world. Organizations have a critical role in protecting the public's health, lowering the danger of transmission, and reestablishing normalcy by putting these measures into place. Following rules, laws, and best practices not only safeguards people's health but also helps the larger international effort to resolve this extraordinary health issue.

1.2 Statement of the Problem

The issue at hand is the pressing requirement for an effective occupancy counter or monitoring system to handle the issues brought on by the COVID-19 epidemic. The following major problems are presented by the current situation:

1. Insufficient monitoring capabilities: Many firms lack a reliable way to precisely track room occupancy. This shortcoming makes it difficult for them to enforce occupancy limits and maintain adherence to physical distance requirements, which could lead to overcrowding and higher transmission risks.

2. Limitations of manual counting: Conventional techniques of manually counting occupants or depending on sign-in sheets are timeconsuming, prone to inaccuracy, and only provide a small amount of real-time information. These manual methods are ineffective, especially in big or active places, making it difficult to efficiently monitor crowd sizes and react quickly to possible threats.

It is essential to create a monitoring system or occupancy counter that offers precise and up-to-date information for various organizations and situations in order to effectively manage these difficulties.

1.3 Objective of the Project

This project's goal is to create a system for tracking and counting room occupancy in real-time that makes use of object detection methods with OpenCV and YOLOv5. The project's precise objectives are as follows:

- Create an accurate and efficient object detection model by implementing the cutting-edge YOLOv5 algorithm with C++ and OpenCV. To enable the model to detect and identify humans within a particular location, with the use of a pretrained dataset by YOLOV5 and filtering only to detect persons.
- Real-time room occupancy monitoring: Make use of the created object detection model to provide real-time room occupancy monitoring. In order to offer real-time updates on the occupancy rate, the system should be able to process video feeds or image streams from cameras positioned inside the room.
- Set specified occupancy limits for the monitored room and enforce them to ensure compliance. Implement systems to monitor the number of occupants the system detects and compare it to the predetermined limitations. When the occupancy limit is achieved or exceeded, alert notifications should be provided to enable prompt enforcement of the physical distance rules.

By meeting these goals, the project hopes to offer a workable and efficient remedy for tracking and counting room occupancy in real-time. In the context of the COVID-19 pandemic and beyond, the technology will assist organizations in enforcing occupancy limitations, upholding physical distancing regulations, and improving all-around safety.

1.4 Scope and Limitation

Scope:

The scope of this project is to create an object detection system for real-time room occupancy monitoring and counting that uses OpenCV and YOLOv5. The following aspects will receive the system's attention:

1. Object detection model: To find and identify people in a room, an object detection model using the YOLOv5 algorithm with OpenCV and C++ is used and utilized.

2. Real-time monitoring entails setting up a system to analyze streaming video or live image feeds from cameras positioned inside the space and deliver real-time updates on the number of occupants.

3. Enforcement of occupancy limits: This feature allows the system to assess the observed occupancy count against predetermined thresholds and to send out alerts when the threshold is met or exceeded.

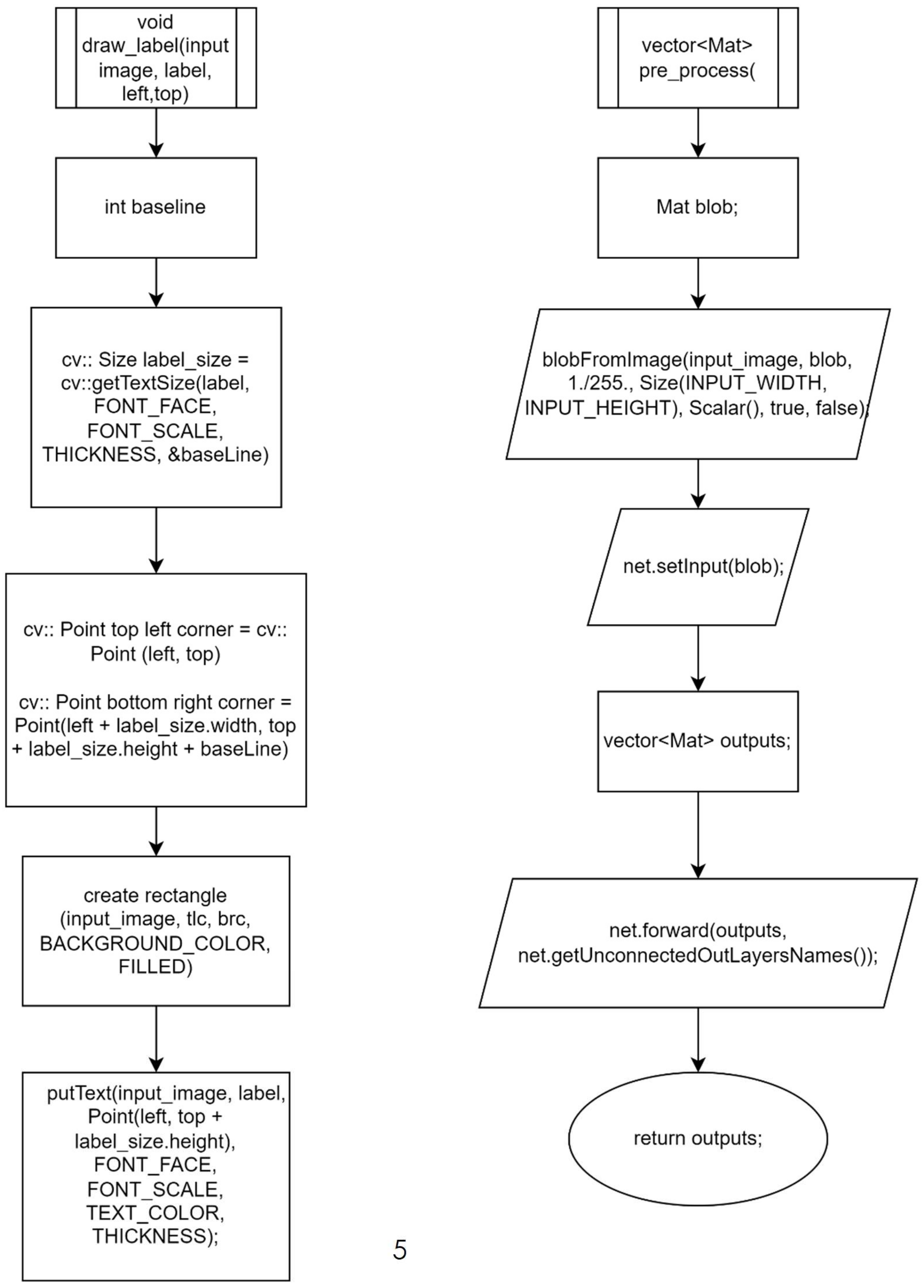
Limitations:

Despite its scale, the project has a few limitations, such as:

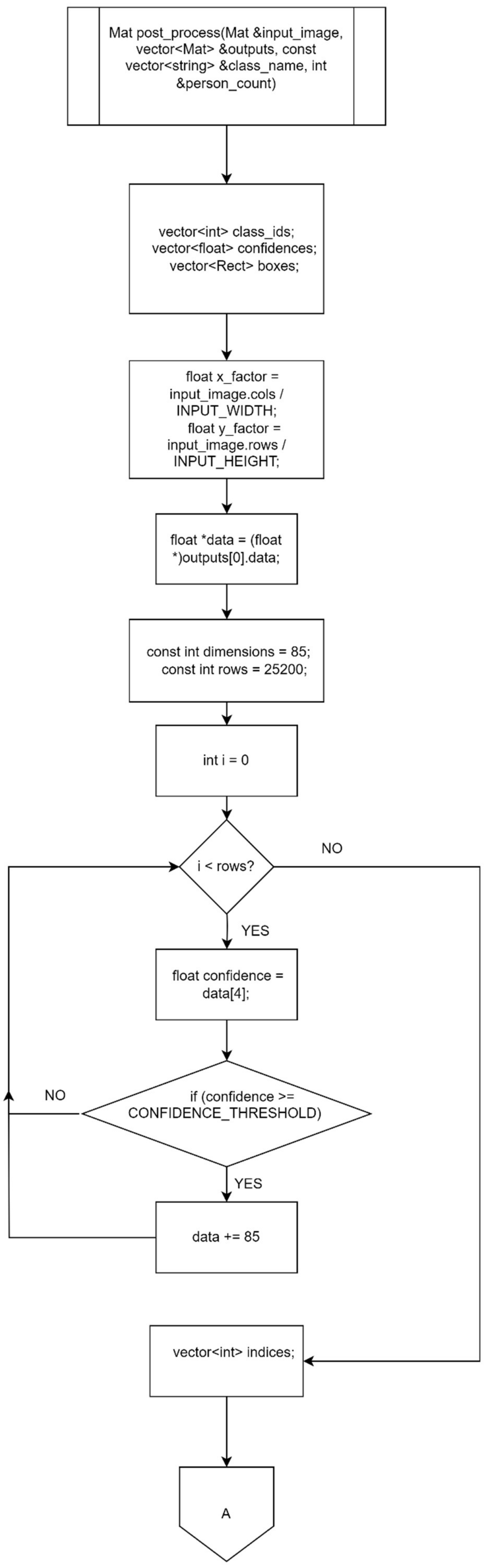
1. The research will largely concentrate on a single camera configuration for tracking room occupancy. It is outside the purview of this project to test the system with several cameras and ensure seamless integration of data from various camera viewpoints.
2. Hardware restrictions: The project will be created based on the resources and capabilities of the already available hardware. The performance and scalability of the system may be impacted by restrictions on camera resolution, processor power, and network bandwidth.
3. Graphical User Interface (GUI): The project seeks to create a userfriendly interface, but due to time and resource limitations, the GUI's design and sophistication may be constrained.
4. Environmental factors: The project will be predicated on ideal environmental circumstances, such as constant lighting and few obstructions. The immediate focus of this project does not include dealing with difficult lighting, occlusions brought on by items or furniture, or other environmental issues.

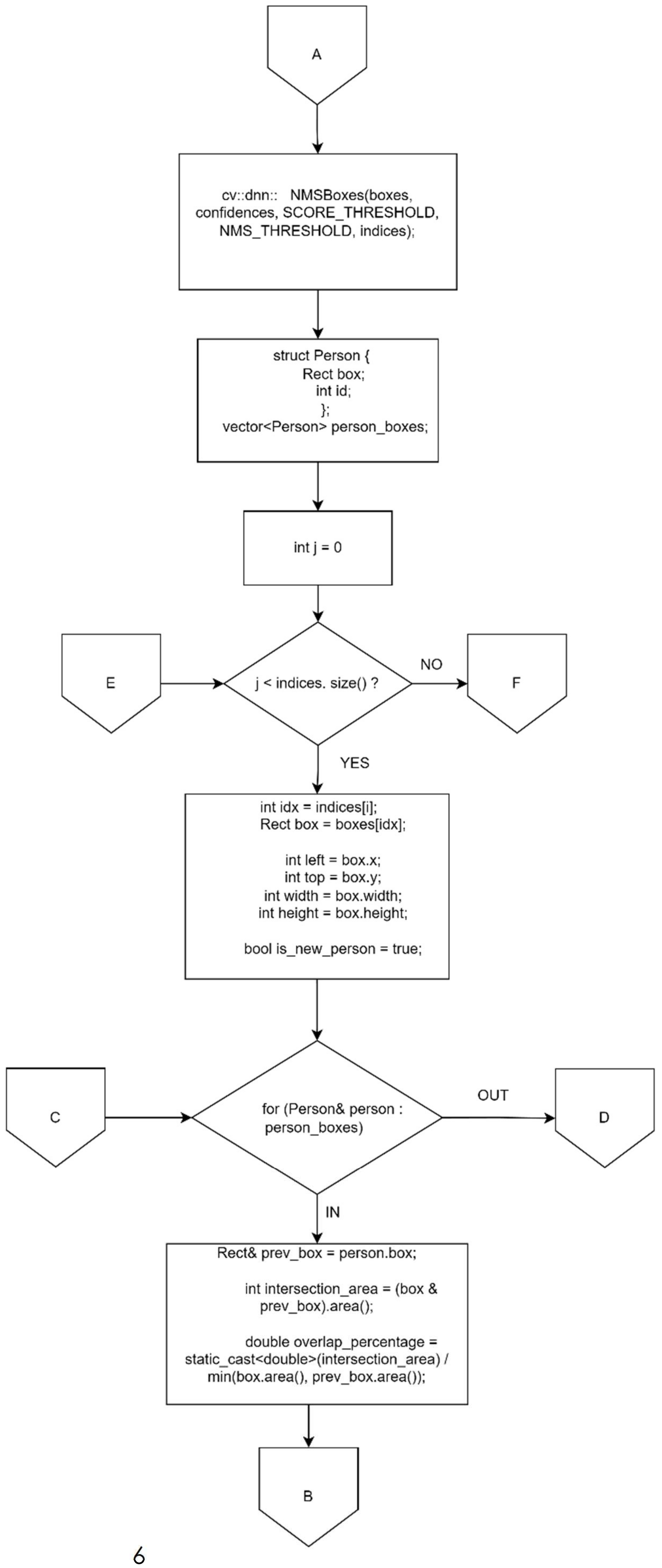
CHAPTER I

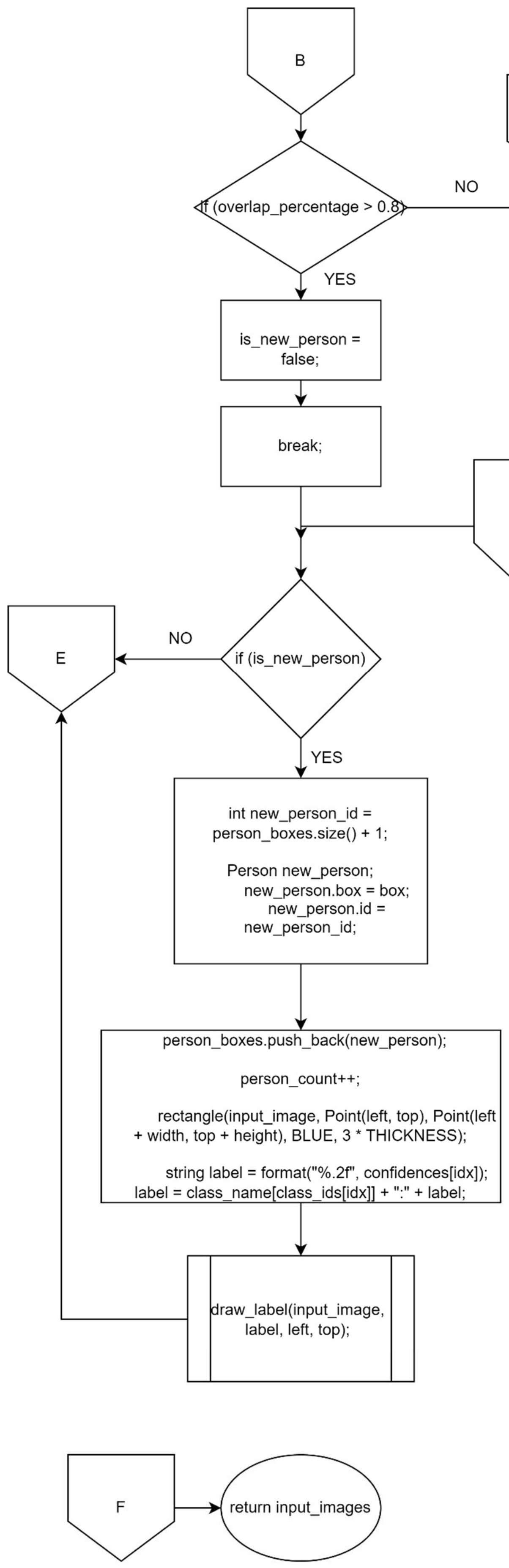
FLOWCHART

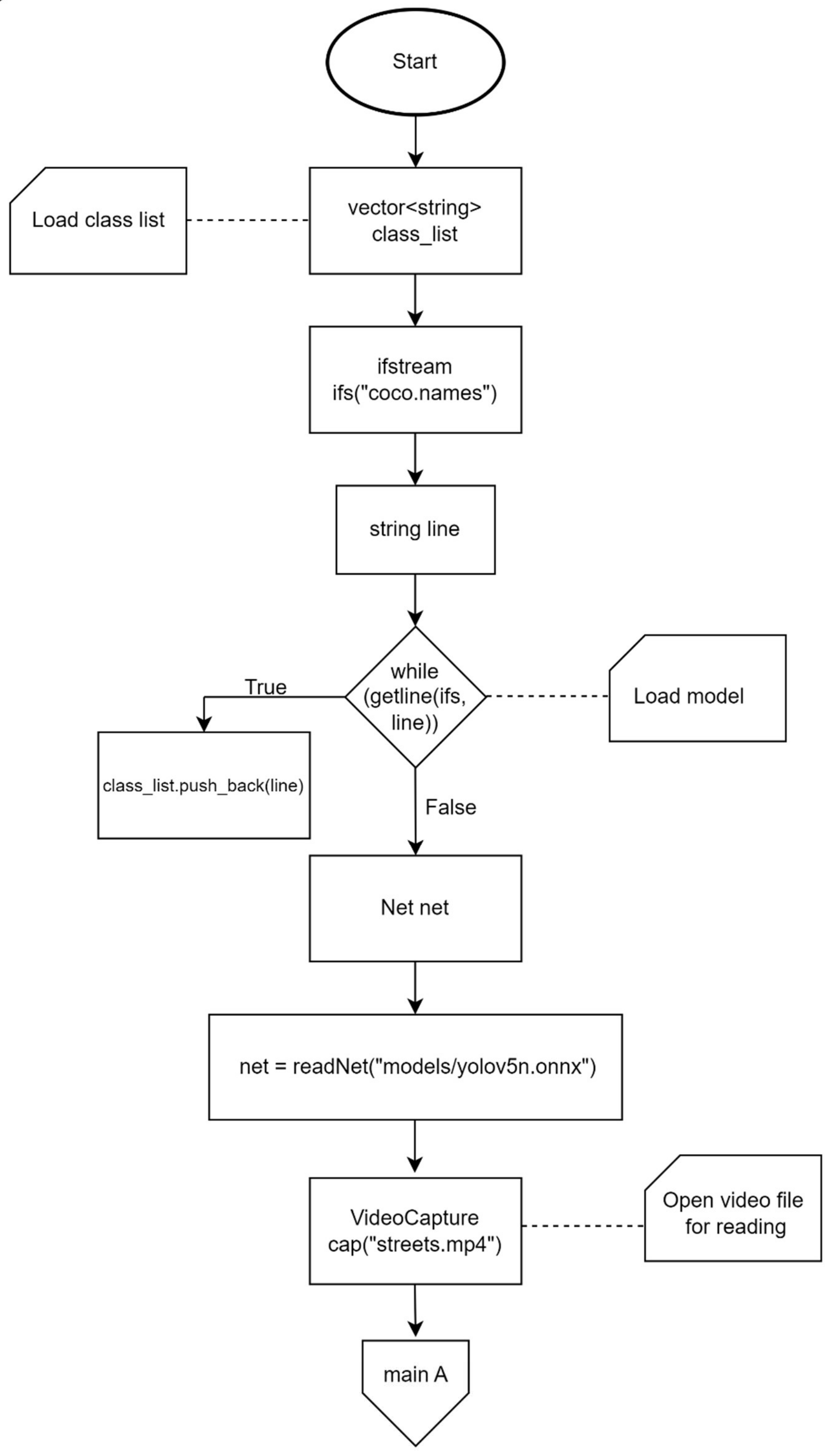


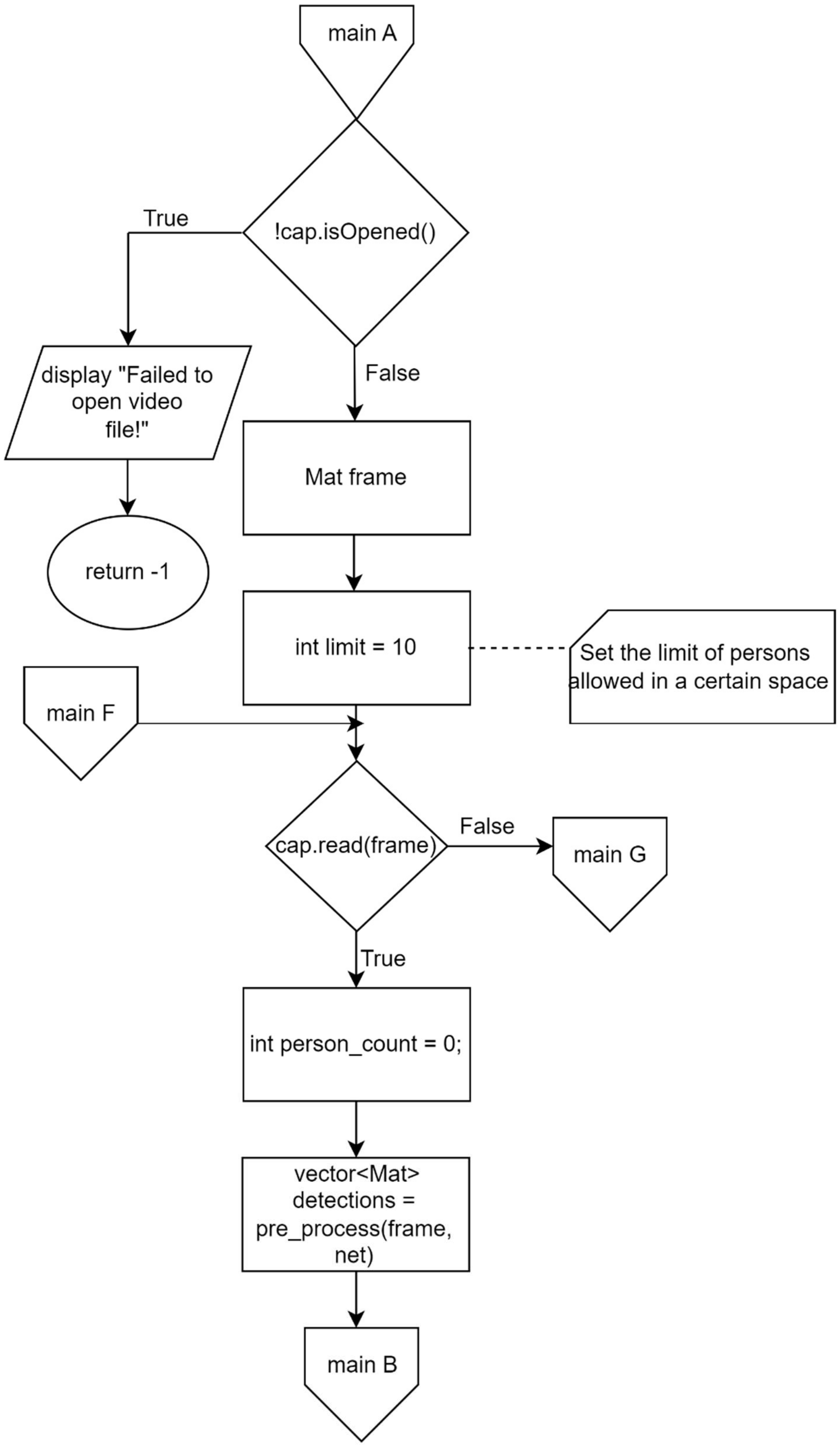
void draw_label(input image, label, left,top)
int baseline
cv:: Size label_size = cv::getTextSize(label, FONT_FACE, FONT_SCALE, THICKNESS, &baseLine)
cv:: Point top left corner = cv:: Point (left, top)
cv:: Point bottom right corner = Point(left + label_size.width, top + label_size.height + baseLine)
create rectangle (input_image, tlc, brc, BACKGROUND_COLOR, FILLED)
putText(input_image, label, Point(left, top + label_size.height), FONT_FACE, FONT_SCALE, TEXT_COLOR, THICKNESS);
vector<Mat> pre_process(
Mat blob;
blobFromImage(input_image, blob, 1./255., Size(INPUT_WIDTH, INPUT_HEIGHT), Scalar(), true, false);
net.setInput(blob);
vector<Mat> outputs;
net.forward(outputs, net.getUnconnectedOutLayersNames());
return outputs;

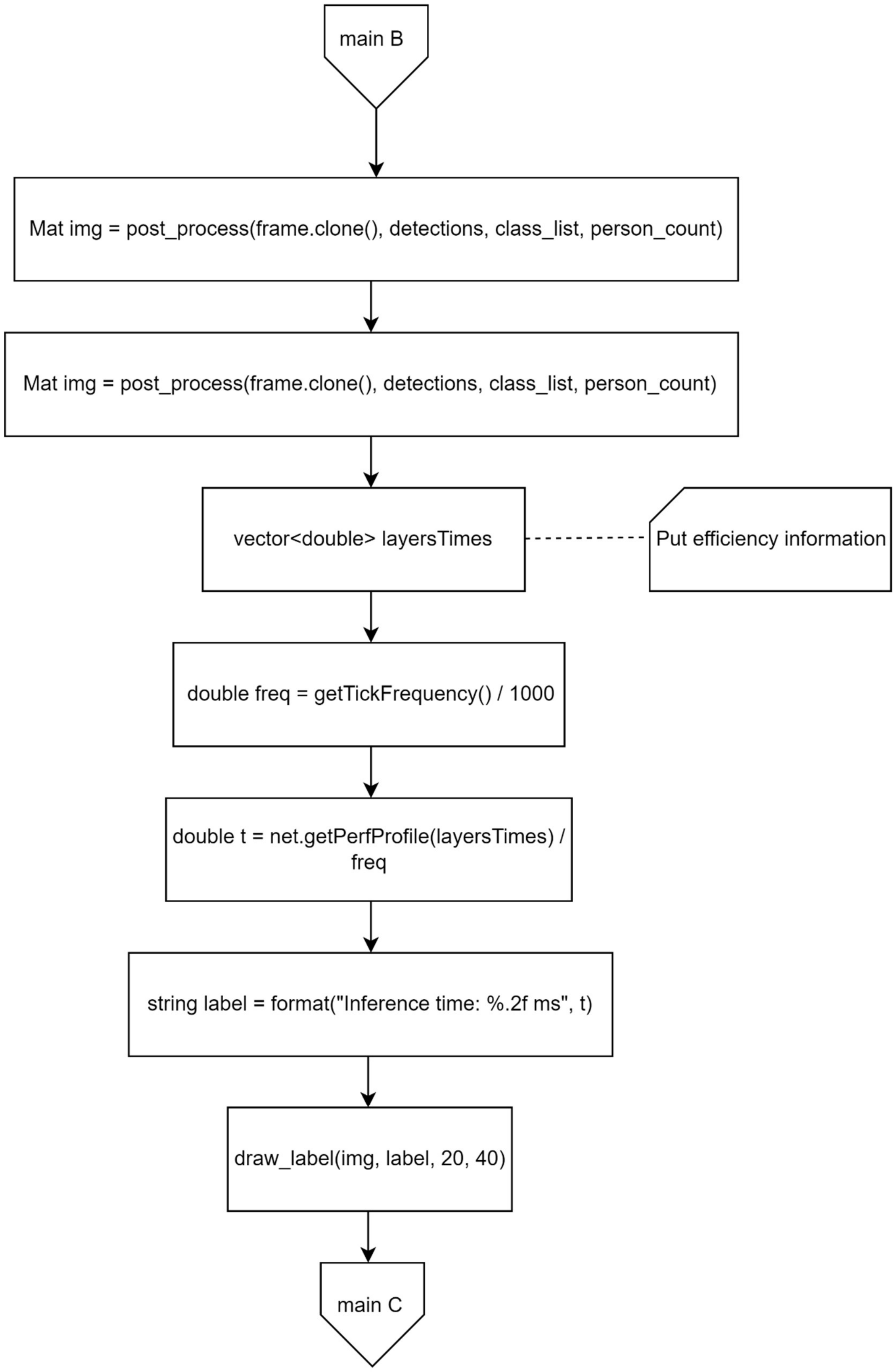

Mat post_process(Mat &input_image, vector<Mat> &outputs, const vector<string> &class_name, int &person_count)
vector<int> class_ids; vector<float> confidences; vector<Rect> boxes;
float x_factor = input_image.cols / INPUT_WIDTH; float y_factor = input_image.rows / INPUT_HEIGHT;
float *data = (float *)outputs[0].data;
const int dimensions = 85; const int rows = 25200;
int i = 0
i < rows?
NO
YES
float confidence = data[4];
if (confidence >= CONFIDENCE_THRESHOLD)
NO
YES
data += 85
vector<int> indices;
A

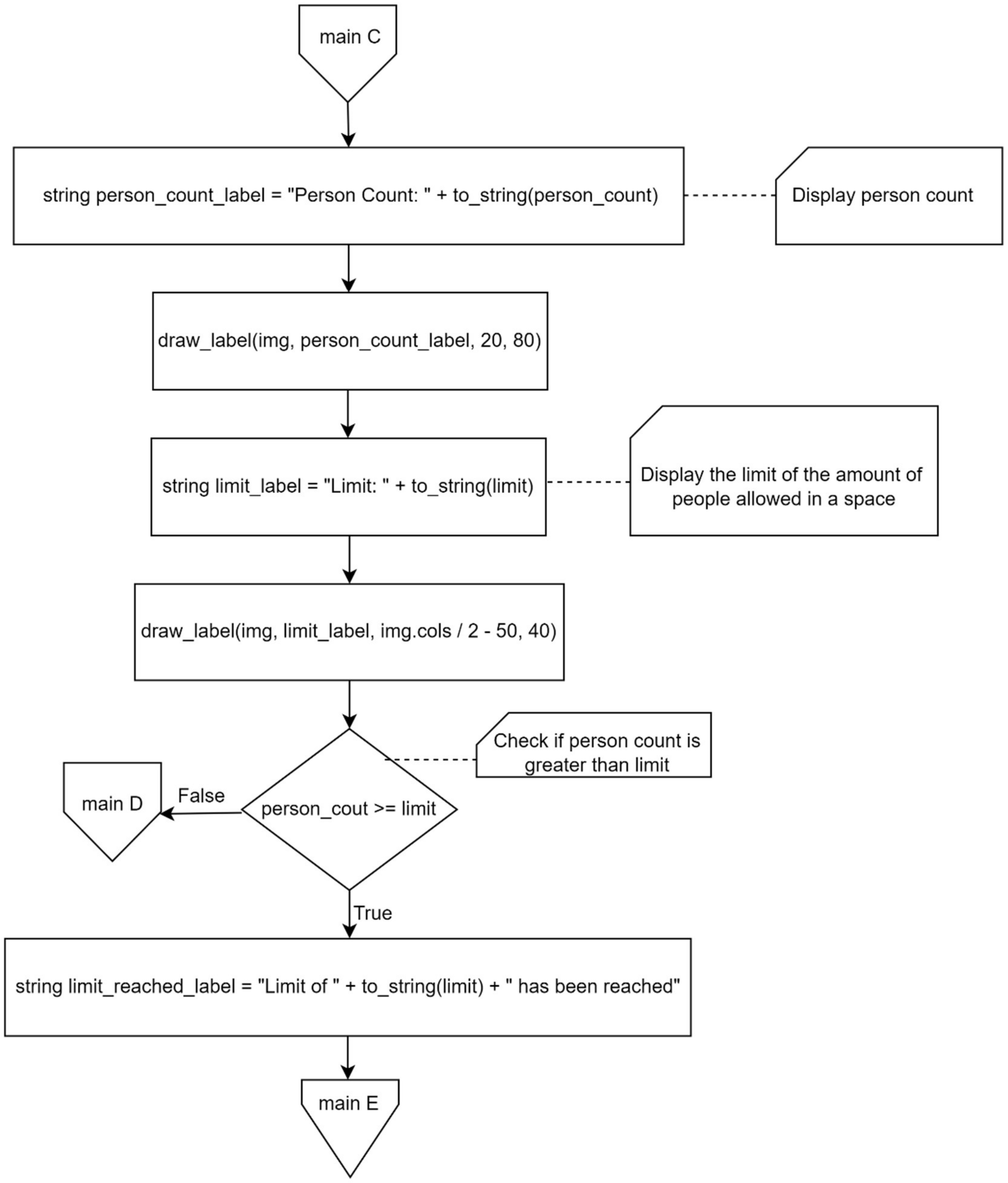

A
cv::dnn:: NMSBoxes(boxes, confidences, SCORE_THRESHOLD, NMS_THRESHOLD, indices);
struct Person { Rect box; int id; }; vector<Person> person_boxes;
int j = 0
E
j < indices. size() ?
NO
F
YES
int idx = indices[i]; Rect box = boxes[idx]; int left = box.x; int top = box.y; int width = box.width; int height = box.height; bool is_new_person = true;
C
for (Person& person : person_boxes)
OUT
D
IN
Rect& prev_box = person.box; int intersection_area = (box & prev_box).area(); double overlap_percentage = static_cast<double>(intersection_area) / min(box.area(), prev_box.area());
B

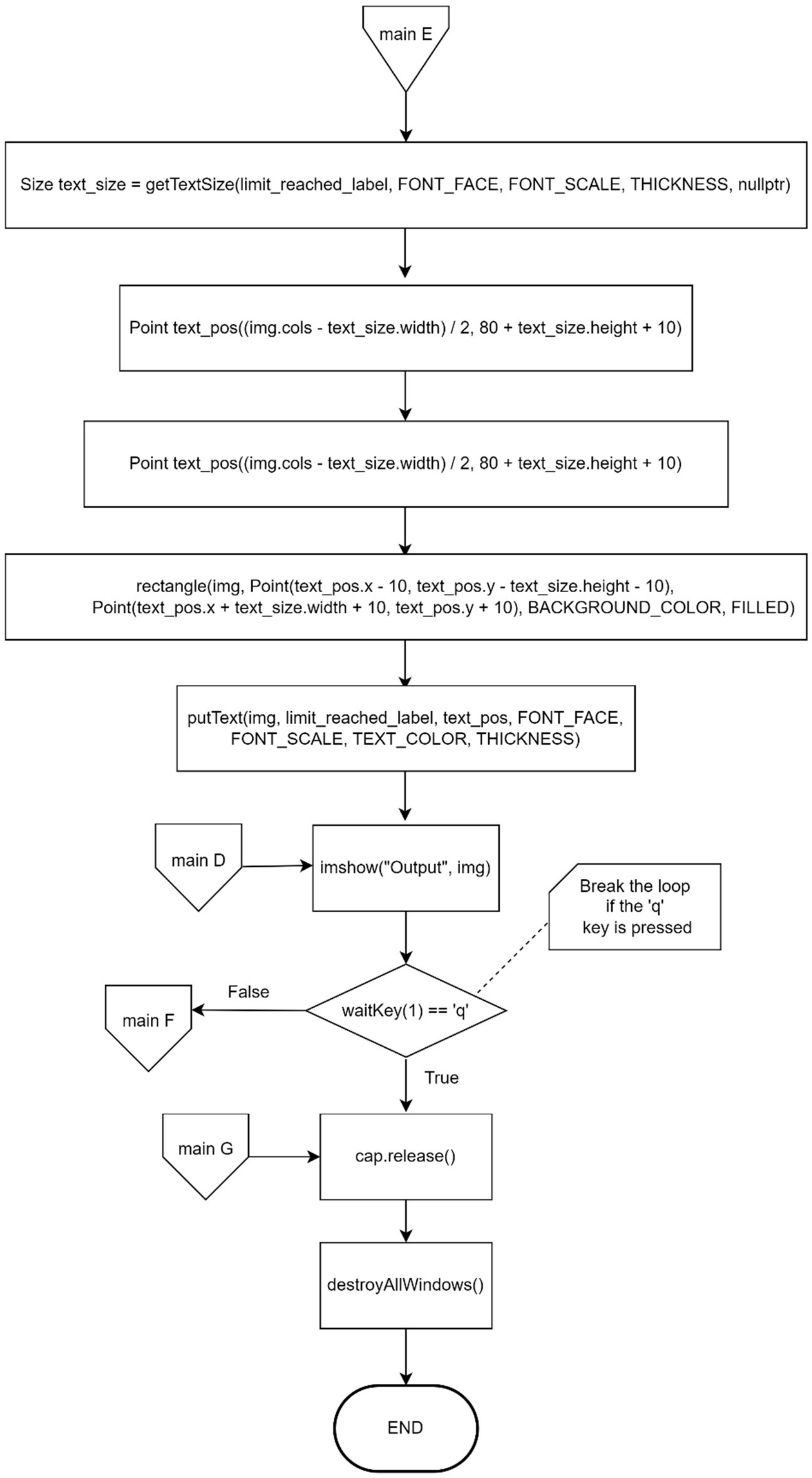

B
if (overlap_percentage > 0.8)
NO
C
YES
is_new_person = false;
break;
D
if (is_new_person)
NO
E
YES
int new_person_id = person_boxes.size() + 1;
Person new_person;
new_person.box = box;
new_person.id = new_person_id;
person_boxes.push_back(new_person);
person_count++;
rectangle(input_image, Point(left, top), Point(left + width, top + height), BLUE, 3 * THICKNESS);
string label = format("%.2f", confidences[idx]);
label = class_name[class_ids[idx]] + ":" + label;
draw_label(input_image, label, left, top);
F
return input_images


Start
Load class list
vector<string> class_list
ifstream ifs("coco.names")
string line
while (getline(ifs, line))
True
class_list.push_back(line)
Load model
False
Net net
net = readNet("models/yolov5n.onnx")
VideoCapture cap("streets.mp4")
Open video file for reading
main A


main A
!cap.isOpened()
True
False
display "Failed to open video file!"
return -1
Mat frame
int limit = 10
Set the limit of persons allowed in a certain space
main F
cap.read(frame)
False
main G
True
int person_count = 0;
vector<Mat> detections = pre_process(frame, net)
main B


main B
Mat img = post_process(frame.clone(), detections, class_list, person_count)
Mat img = post_process(frame.clone(), detections, class_list, person_count)
vector<double> layersTimes
Put efficiency information
double freq = getTickFrequency() / 1000
double t = net.getPerfProfile(layersTimes) / freq
string label = format("Inference time: %.2f ms", t)
draw_label(img, label, 20, 40)
main C


main C
string person_count_label = "Person Count: " + to_string(person_count)
Display person count
draw_label(img, person_count_label, 20, 80)
string limit_label = "Limit: " + to_string(limit)
Display the limit of the amount of people allowed in a space
draw_label(img, limit_label, img.cols / 2 - 50, 40)
Check if person count is greater than limit
person_cout >= limit
False
main D
True
string limit_reached_label = "Limit of " + to_string(limit) + " has been reached"
main E


main E
Size text_size = getTextSize(limit_reached_label, FONT_FACE, FONT_SCALE, THICKNESS, nullptr)
Point text_pos((img.cols - text_size.width) / 2, 80 + text_size.height + 10)
Point text_pos((img.cols - text_size.width) / 2, 80 + text_size.height + 10)
rectangle(img, Point(text_pos.x - 10, text_pos.y - text_size.height - 10), Point(text_pos.x + text_size.width + 10, text_pos.y + 10), BACKGROUND_COLOR, FILLED)
putText(img, limit_reached_label, text_pos, FONT_FACE, FONT_SCALE, TEXT_COLOR, THICKNESS)
main D
imshow("Output", img)
Break the loop if the 'q' key is pressed
False
main F
waitKey(1) == 'q'
True
main G
cap.release()
destroyAllWindows()
END

Flowchart explanation

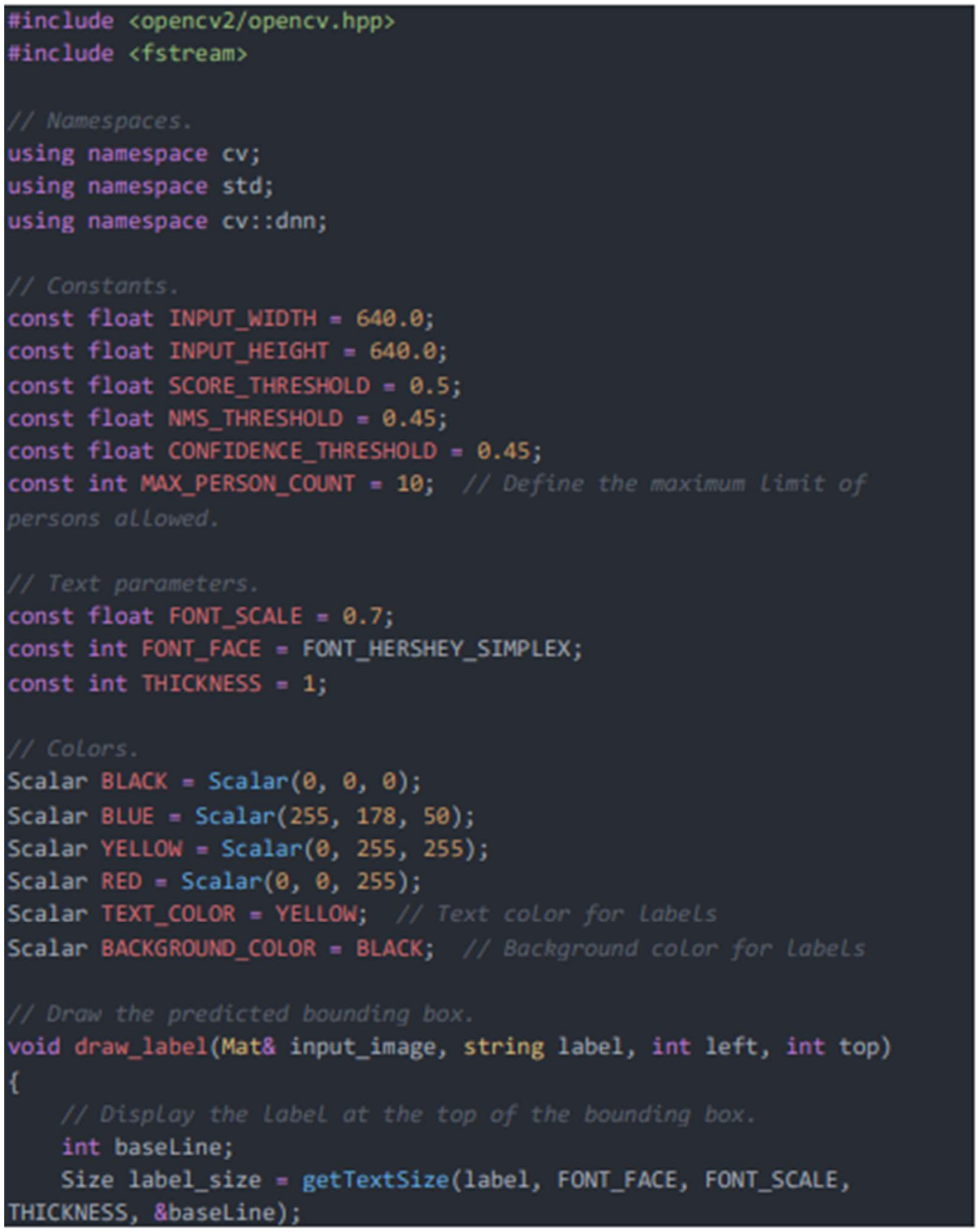
2.1 The **draw_label** function takes an input image, a label string, and the coordinates (left, top) where the label should be placed. It calculates the size of the label text, adjusts the top coordinate to ensure the label is displayed at the top of the bounding box, and then draws a black rectangle at the specified coordinates. The function puts the label text on the black rectangle using putText. This function is used to draw labels on bounding boxes in the object detection process.

2.2 The **pre_process** function takes an input image and a neural network model (Net) as parameters. It converts the image to a blob, which is a preprocessed form suitable for input to the neural network. The function sets the input of the neural network to the blob and performs forward propagation to obtain the outputs. These outputs are returned as a vector of Mat objects. This function prepares the input image for object detection by converting it to a format suitable for the neural network model.

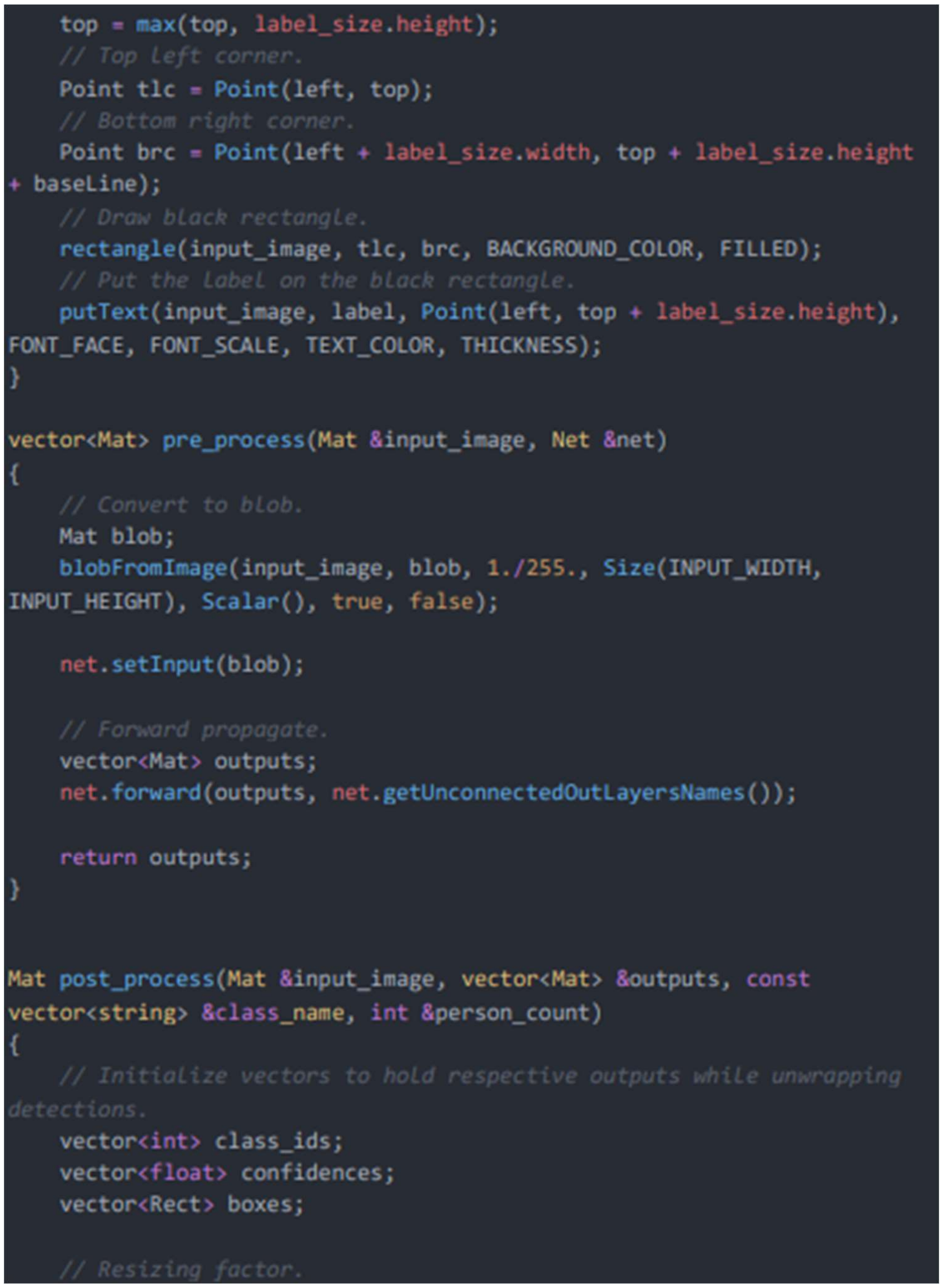
2.3 The **post_process** function takes the input image, the outputs of the neural network, a vector of class names, and a person count as parameters. It processes the outputs to extract bounding box coordinates, class labels, and confidences. It performs non-maximum suppression to remove duplicate detections and overlapping bounding boxes. It assigns unique IDs to individual persons and keeps track of the person count. The function draws bounding boxes and labels on the input image for each detected person. The processed image is returned as the output. This function handles the postprocessing of object detection results and generates the final annotated image with bounding boxes and labels.

2.4 The **main function** serves as the entry point of the program. It starts by loading the list of class names from a file and the neural network model from a pre-trained ONNX file. Then, it opens a video file for reading frame by frame. Inside the main loop, it performs the following steps: pre-processes the frame using the pre_process function to obtain the detection outputs, passes the pre-processed frame and detection outputs to the post_process function to annotate the frame with bounding boxes and labels, displays the annotated frame with additional information such as inference time and person count, and checks if the person count exceeds a specified limit and displays a warning message if it does. The loop continues until the 'q' key is pressed or the video ends. Finally, it releases the video capture and closes any open windows. This main function orchestrates the entire object detection process using the pre_process and post_process functions and handles the input/output operations.

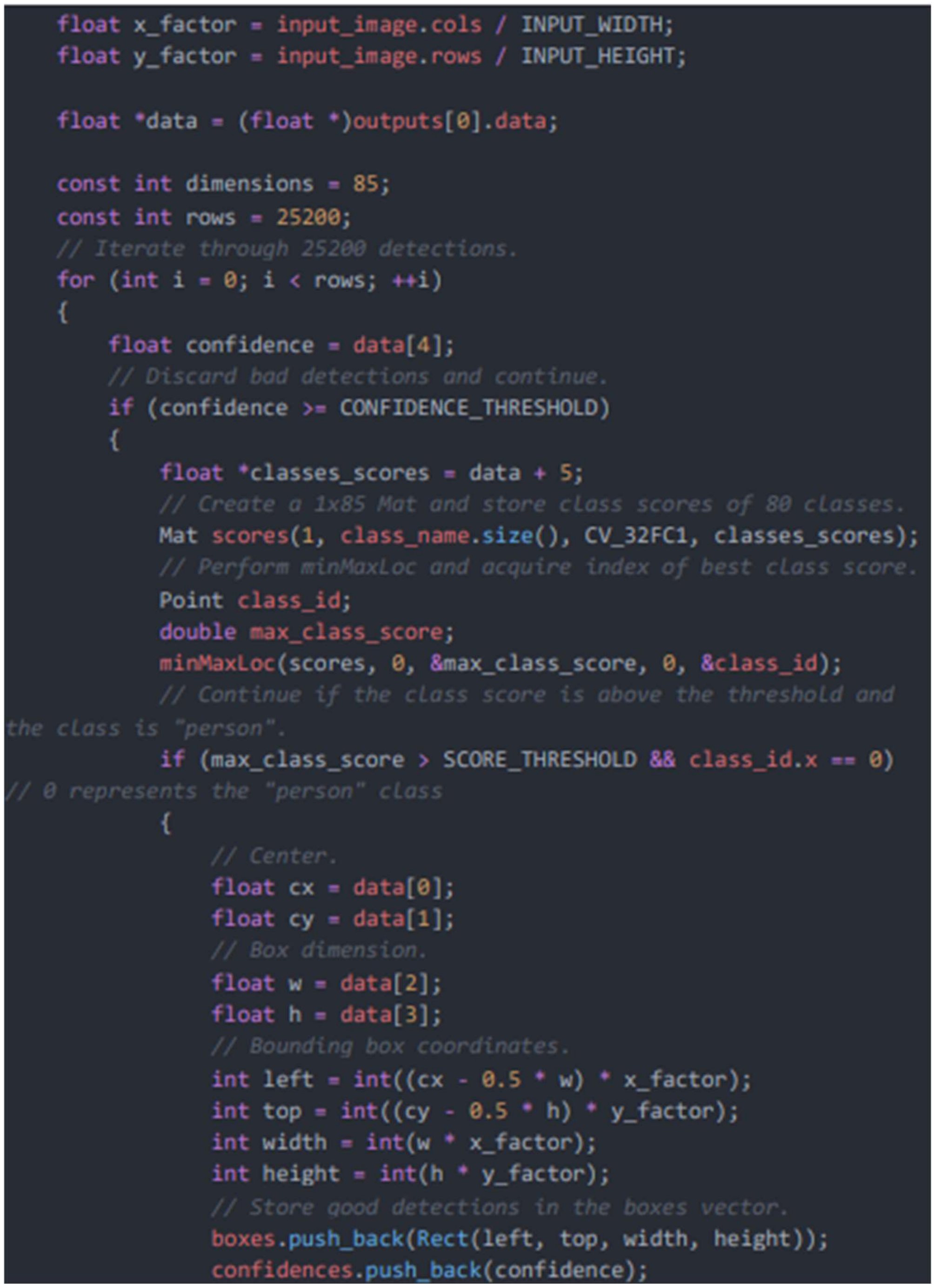
CHAPTER 3
SOURCE CODE



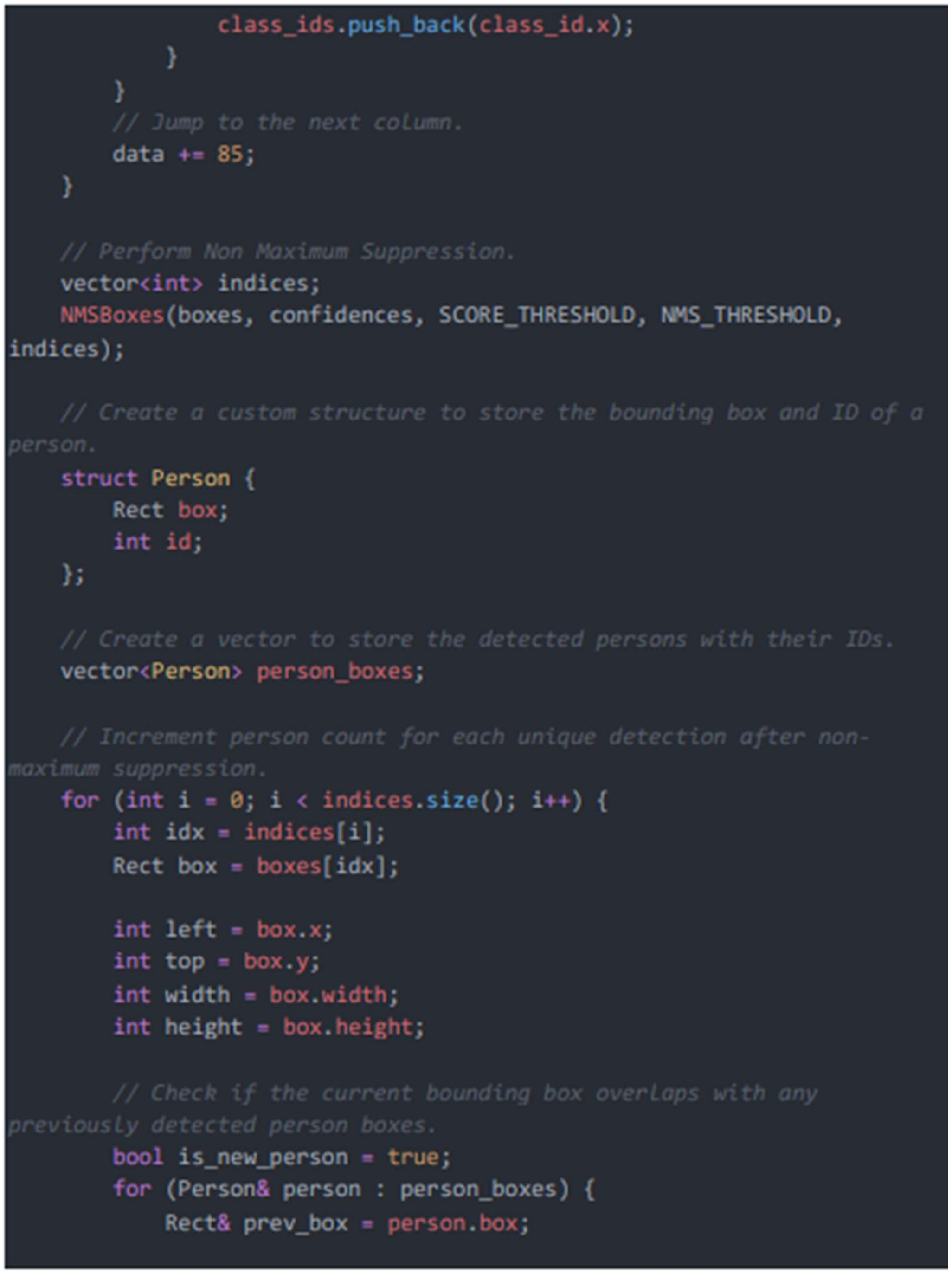
```cpp
#include <opencv2/opencv.hpp>
#include <fstream>

// Namespaces.
using namespace cv;
using namespace std;
using namespace cv::dnn;

// Constants.
const float INPUT_WIDTH = 640.0;
const float INPUT_HEIGHT = 640.0;
const float SCORE_THRESHOLD = 0.5;
const float NMS_THRESHOLD = 0.45;
const float CONFIDENCE_THRESHOLD = 0.45;
const int MAX_PERSON_COUNT = 10;  // Define the maximum limit of
persons allowed.

// Text parameters.
const float FONT_SCALE = 0.7;
const int FONT_FACE = FONT_HERSHEY_SIMPLEX;
const int THICKNESS = 1;

// Colors.
Scalar BLACK = Scalar(0, 0, 0);
Scalar BLUE = Scalar(255, 178, 50);
Scalar YELLOW = Scalar(0, 255, 255);
Scalar RED = Scalar(0, 0, 255);
Scalar TEXT_COLOR = YELLOW;  // Text color for labels
Scalar BACKGROUND_COLOR = BLACK;  // Background color for labels

// Draw the predicted bounding box.
void draw_label(Mat& input_image, string label, int left, int top)
{
    // Display the label at the top of the bounding box.
    int baseLine;
    Size label_size = getTextSize(label, FONT_FACE, FONT_SCALE,
THICKNESS, &baseLine);
}
```



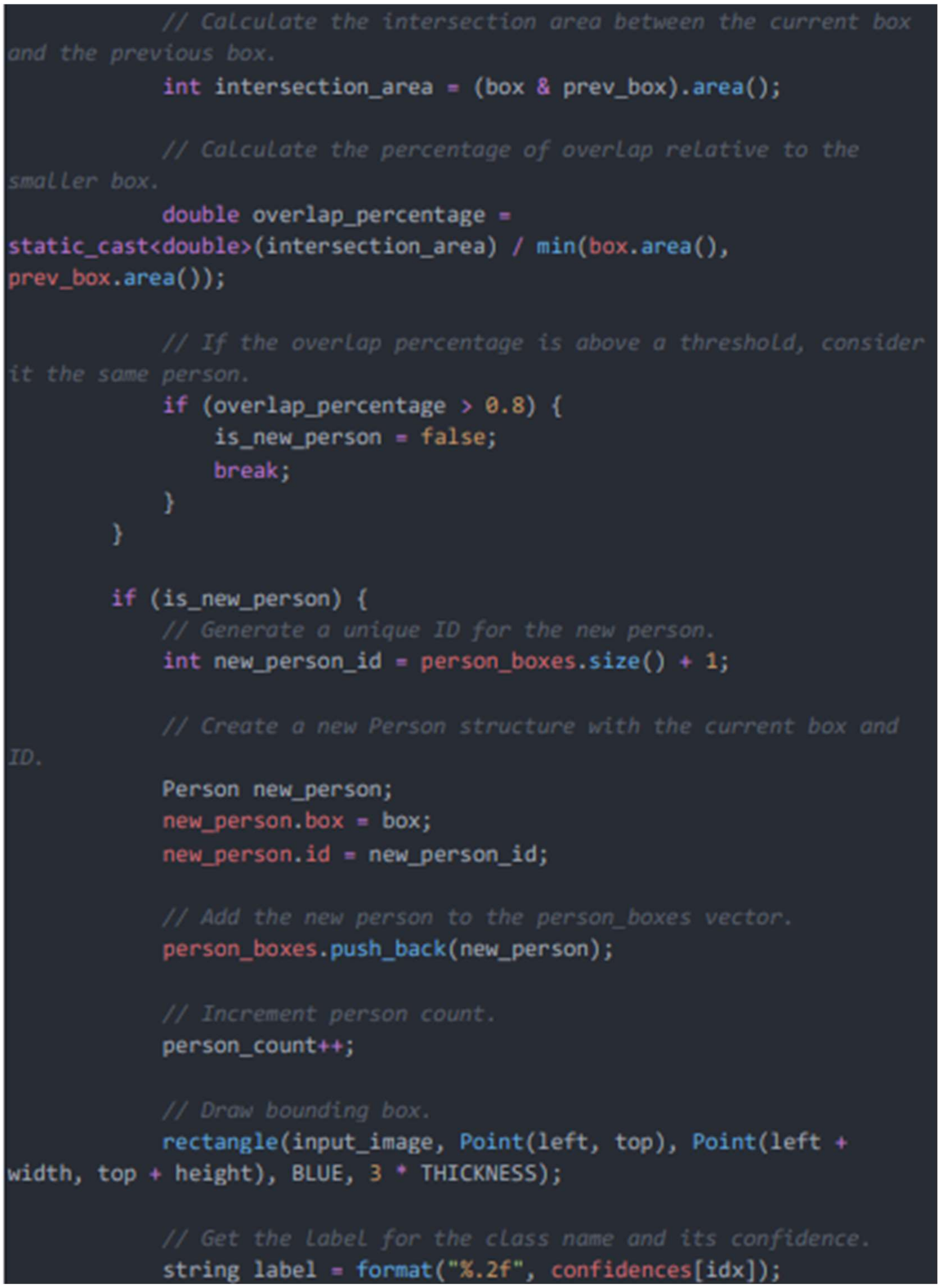
```
    top = max(top, label_size.height);
    // Top left corner.
    Point tlc = Point(left, top);
    // Bottom right corner.
    Point brc = Point(left + label_size.width, top + label_size.height
+ baseLine);
    // Draw black rectangle.
    rectangle(input_image, tlc, brc, BACKGROUND_COLOR, FILLED);
    // Put the label on the black rectangle.
    putText(input_image, label, Point(left, top + label_size.height),
FONT_FACE, FONT_SCALE, TEXT_COLOR, THICKNESS);
}

vector<Mat> pre_process(Mat &input_image, Net &net)
{
    // Convert to blob.
    Mat blob;
    blobFromImage(input_image, blob, 1./255., Size(INPUT_WIDTH,
INPUT_HEIGHT), Scalar(), true, false);

    net.setInput(blob);

    // Forward propagate.
    vector<Mat> outputs;
    net.forward(outputs, net.getUnconnectedOutLayersNames());

    return outputs;
}


Mat post_process(Mat &input_image, vector<Mat> &outputs, const
vector<string> &class_name, int &person_count)
{
    // Initialize vectors to hold respective outputs while unwrapping
detections.
    vector<int> class_ids;
    vector<float> confidences;
    vector<Rect> boxes;

    // Resizing factor.
```



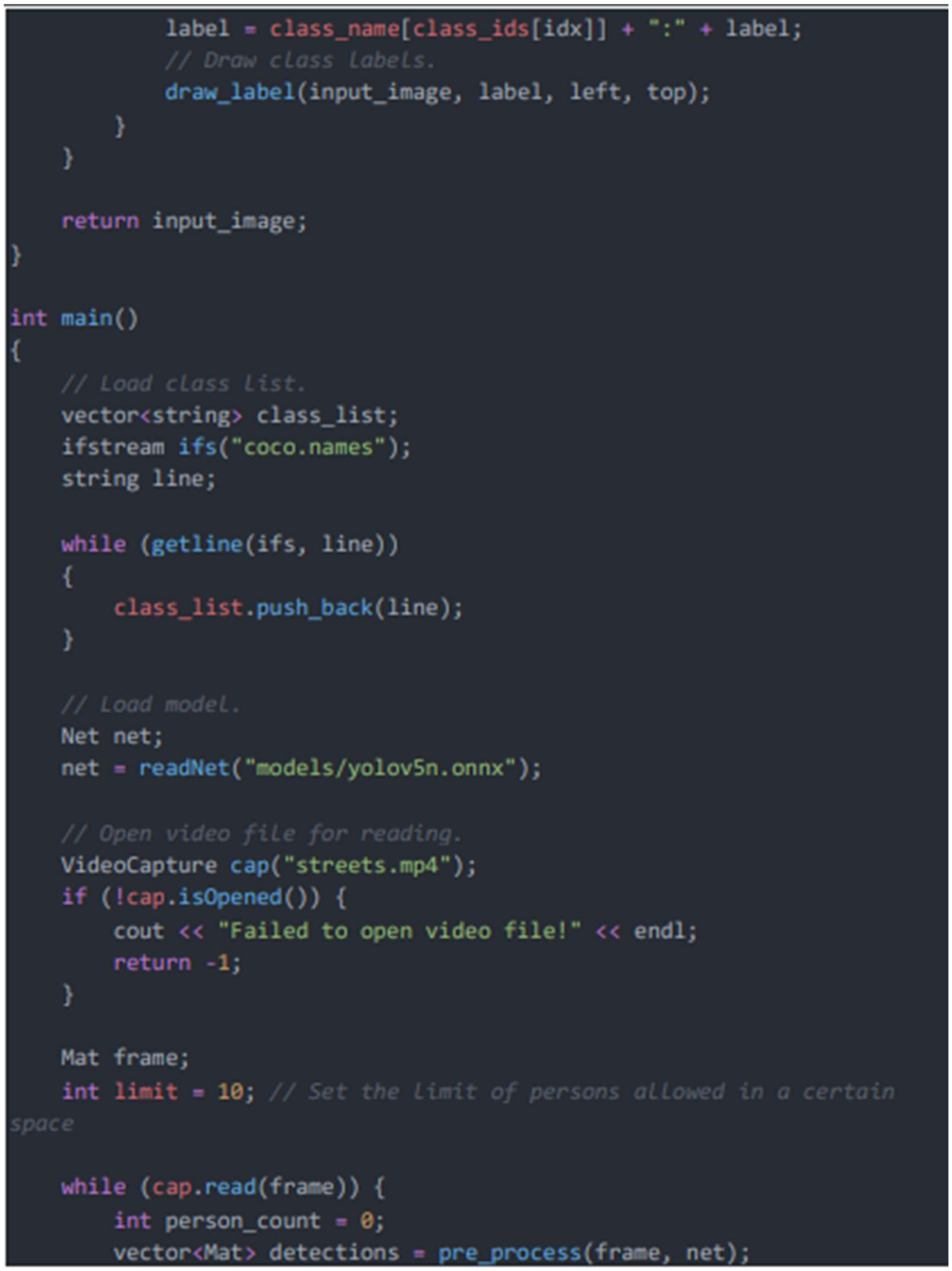
```
    float x_factor = input_image.cols / INPUT_WIDTH;
    float y_factor = input_image.rows / INPUT_HEIGHT;

    float *data = (float *)outputs[0].data;

    const int dimensions = 85;
    const int rows = 25200;
    // Iterate through 25200 detections.
    for (int i = 0; i < rows; ++i)
    {
        float confidence = data[4];
        // Discard bad detections and continue.
        if (confidence >= CONFIDENCE_THRESHOLD)
        {
            float *classes_scores = data + 5;
            // Create a 1x85 Mat and store class scores of 80 classes.
            Mat scores(1, class_name.size(), CV_32FC1, classes_scores);
            // Perform minMaxLoc and acquire index of best class score.
            Point class_id;
            double max_class_score;
            minMaxLoc(scores, 0, &max_class_score, 0, &class_id);
            // Continue if the class score is above the threshold and
the class is "person".
            if (max_class_score > SCORE_THRESHOLD && class_id.x == 0)
// 0 represents the "person" class
            {
                // Center.
                float cx = data[0];
                float cy = data[1];
                // Box dimension.
                float w = data[2];
                float h = data[3];
                // Bounding box coordinates.
                int left = int((cx - 0.5 * w) * x_factor);
                int top = int((cy - 0.5 * h) * y_factor);
                int width = int(w * x_factor);
                int height = int(h * y_factor);
                // Store good detections in the boxes vector.
                boxes.push_back(Rect(left, top, width, height));
                confidences.push_back(confidence);
```



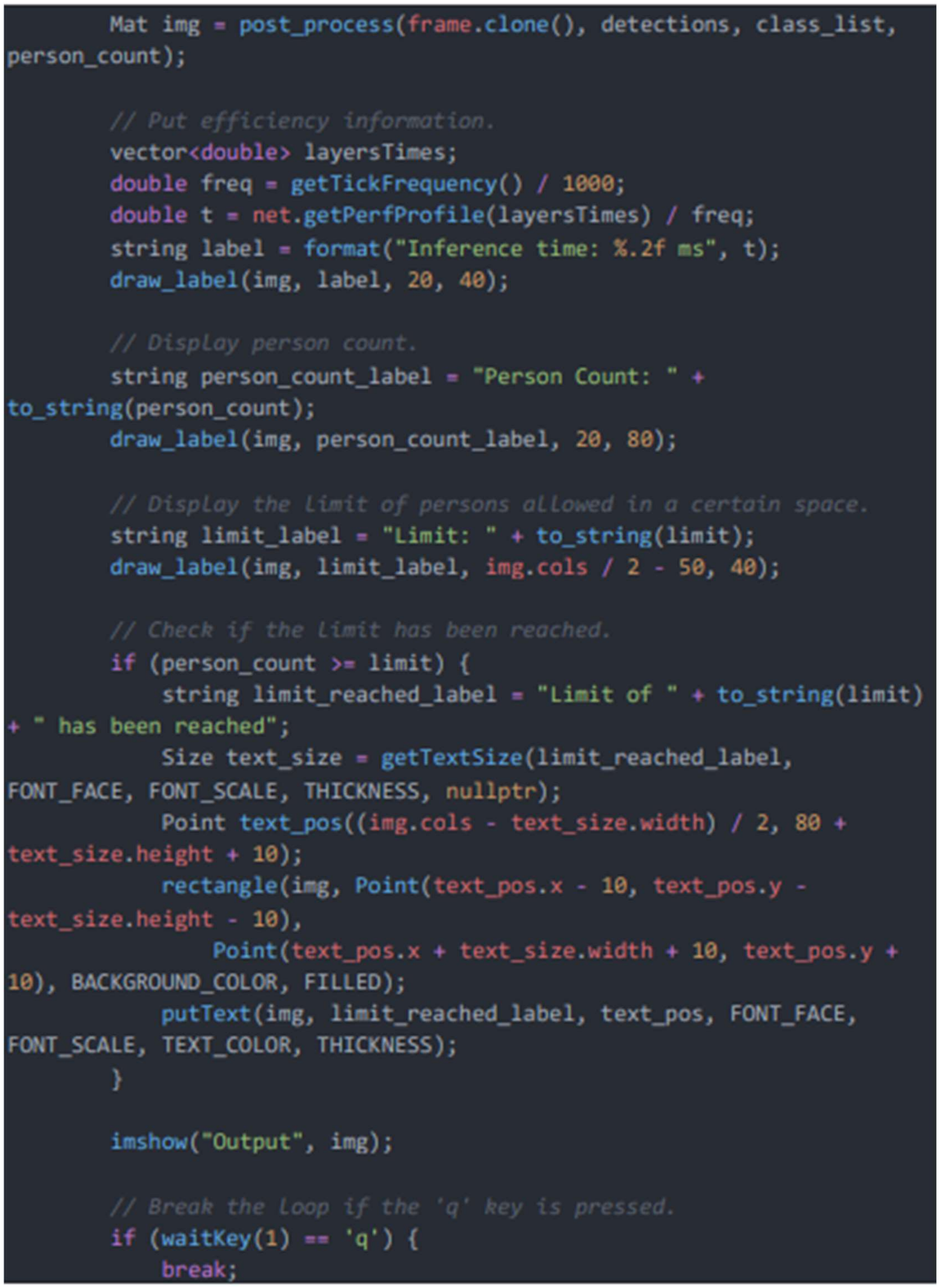
```
                class_ids.push_back(class_id.x);
            }
        }
        // Jump to the next column.
        data += 85;
    }

    // Perform Non Maximum Suppression.
    vector<int> indices;
    NMSBoxes(boxes, confidences, SCORE_THRESHOLD, NMS_THRESHOLD,
indices);

    // Create a custom structure to store the bounding box and ID of a
person.
    struct Person {
        Rect box;
        int id;
    };

    // Create a vector to store the detected persons with their IDs.
    vector<Person> person_boxes;

    // Increment person count for each unique detection after non-
maximum suppression.
    for (int i = 0; i < indices.size(); i++) {
        int idx = indices[i];
        Rect box = boxes[idx];

        int left = box.x;
        int top = box.y;
        int width = box.width;
        int height = box.height;

        // Check if the current bounding box overlaps with any
previously detected person boxes.
        bool is_new_person = true;
        for (Person& person : person_boxes) {
            Rect& prev_box = person.box;
```



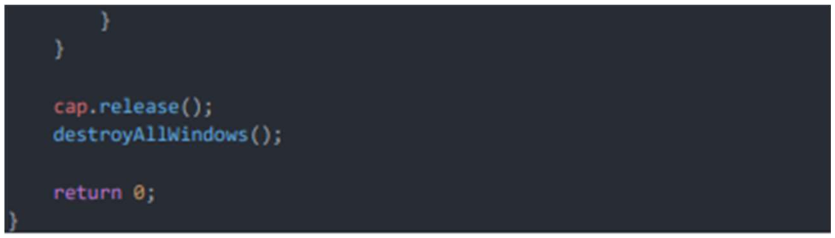
```
            // Calculate the intersection area between the current box
and the previous box.
            int intersection_area = (box & prev_box).area();

            // Calculate the percentage of overlap relative to the
smaller box.
            double overlap_percentage =
static_cast<double>(intersection_area) / min(box.area(),
prev_box.area());

            // If the overlap percentage is above a threshold, consider
it the same person.
            if (overlap_percentage > 0.8) {
                is_new_person = false;
                break;
            }
        }

        if (is_new_person) {
            // Generate a unique ID for the new person.
            int new_person_id = person_boxes.size() + 1;

            // Create a new Person structure with the current box and
ID.
            Person new_person;
            new_person.box = box;
            new_person.id = new_person_id;

            // Add the new person to the person_boxes vector.
            person_boxes.push_back(new_person);

            // Increment person count.
            person_count++;

            // Draw bounding box.
            rectangle(input_image, Point(left, top), Point(left +
width, top + height), BLUE, 3 * THICKNESS);

            // Get the label for the class name and its confidence.
            string label = format("%.2f", confidences[idx]);
```



```
            label = class_name[class_ids[idx]] + ":" + label;
            // Draw class labels.
            draw_label(input_image, label, left, top);
        }
    }

    return input_image;
}

int main()
{
    // Load class list.
    vector<string> class_list;
    ifstream ifs("coco.names");
    string line;

    while (getline(ifs, line))
    {
        class_list.push_back(line);
    }

    // Load model.
    Net net;
    net = readNet("models/yolov5n.onnx");

    // Open video file for reading.
    VideoCapture cap("streets.mp4");
    if (!cap.isOpened()) {
        cout << "Failed to open video file!" << endl;
        return -1;
    }

    Mat frame;
    int limit = 10; // Set the limit of persons allowed in a certain
space

    while (cap.read(frame)) {
        int person_count = 0;
        vector<Mat> detections = pre_process(frame, net);
```



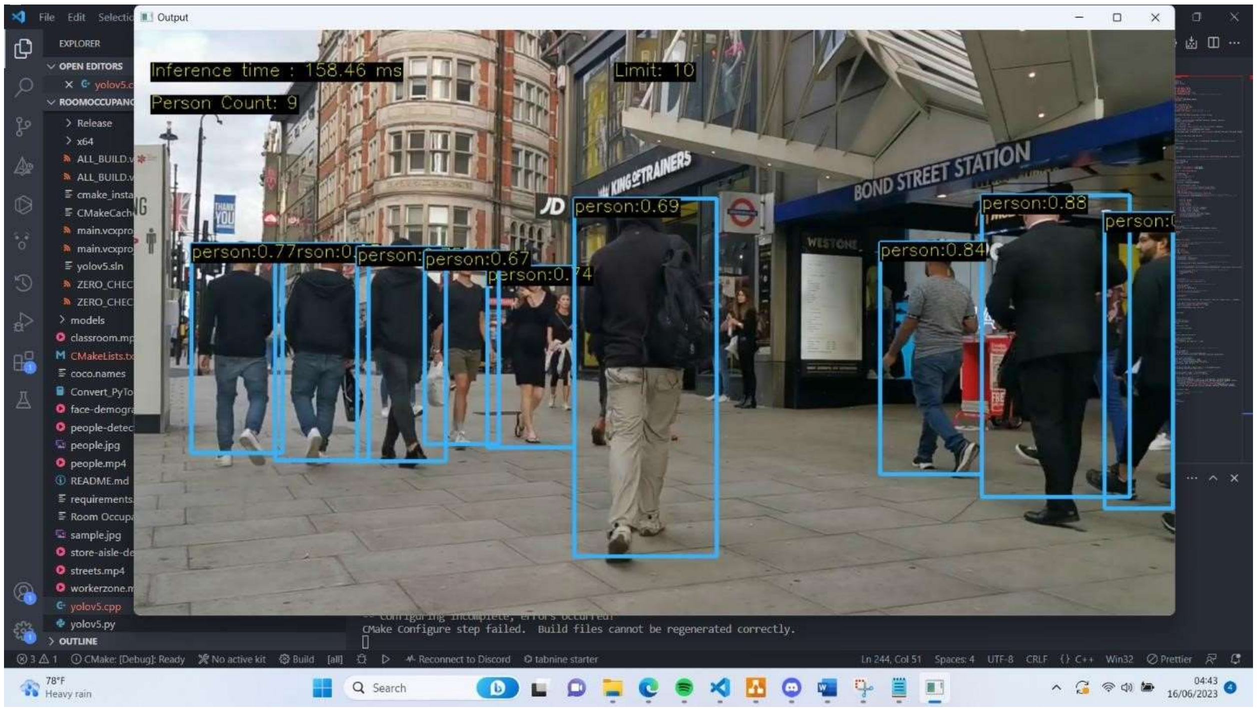
```
        Mat img = post_process(frame.clone(), detections, class_list,
person_count);

        // Put efficiency information.
        vector<double> layersTimes;
        double freq = getTickFrequency() / 1000;
        double t = net.getPerfProfile(layersTimes) / freq;
        string label = format("Inference time: %.2f ms", t);
        draw_label(img, label, 20, 40);

        // Display person count.
        string person_count_label = "Person Count: " +
to_string(person_count);
        draw_label(img, person_count_label, 20, 80);

        // Display the limit of persons allowed in a certain space.
        string limit_label = "Limit: " + to_string(limit);
        draw_label(img, limit_label, img.cols / 2 - 50, 40);

        // Check if the limit has been reached.
        if (person_count >= limit) {
            string limit_reached_label = "Limit of " + to_string(limit)
+ " has been reached";
            Size text_size = getTextSize(limit_reached_label,
FONT_FACE, FONT_SCALE, THICKNESS, nullptr);
            Point text_pos((img.cols - text_size.width) / 2, 80 +
text_size.height + 10);
            rectangle(img, Point(text_pos.x - 10, text_pos.y -
text_size.height - 10),
                Point(text_pos.x + text_size.width + 10, text_pos.y +
10), BACKGROUND_COLOR, FILLED);
            putText(img, limit_reached_label, text_pos, FONT_FACE,
FONT_SCALE, TEXT_COLOR, THICKNESS);
        }

        imshow("Output", img);

        // Break the loop if the 'q' key is pressed.
        if (waitKey(1) == 'q') {
            break;
```



```
        }
    }

    cap.release();
    destroyAllWindows();

    return 0;
}
```

CHAPTER 4 SAMPLE USER INTERFACE

Example 4.1

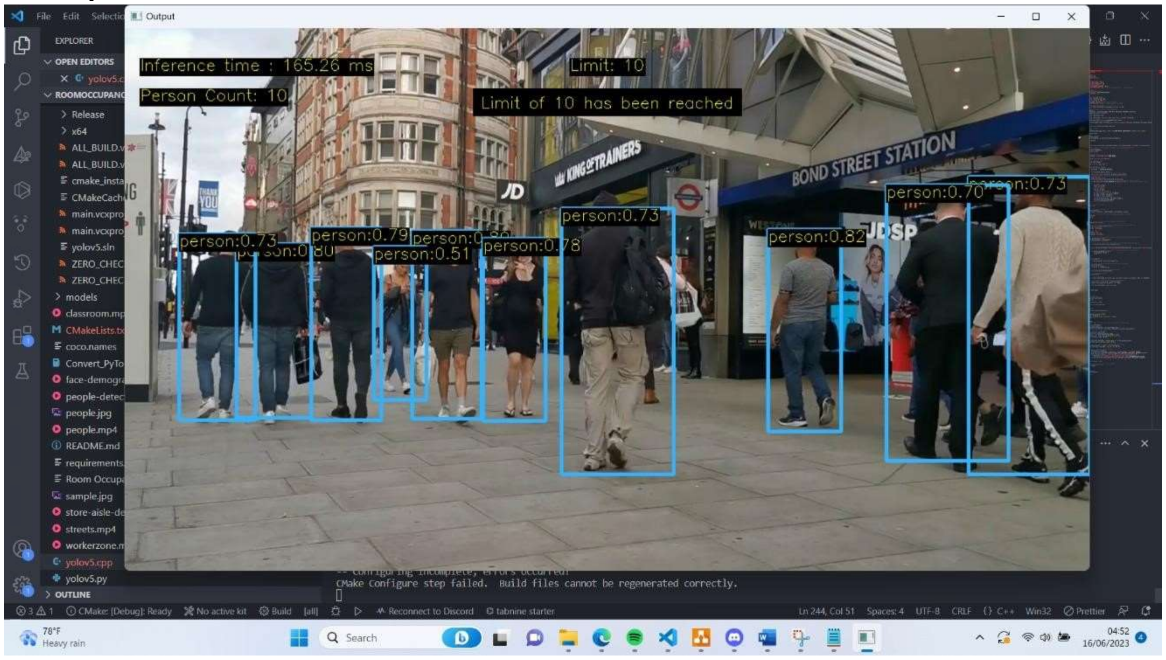


```
Inference time : 158.46 ms
Person Count: 9
Limit: 10
```

```
CMake Configure step failed.  Build files cannot be regenerated correctly.
```

A sample output with set limit to 10. The person counter is currently 9 and is not greater than or equal to the limit. Thus, no warning has appeared in the output in this example. If the amount of people detected will be greater than or equal to 10, a warning message will appear, as shown in example 4.2

Example 4.2



```
Inference time : 165.26 ms
Person Count: 10
Limit: 10
Limit of 10 has been reached
```

```
CMake Configure step failed.  Build files cannot be regenerated correctly.
```

Here in this output, the person count has detected 10 people. Thus, the person count has now been updated to 10. The person count is now greater than or equal to 10. The limit has now been reached. Because of that, the program will display that the limit of 10 has been reached.