Number Plate Detection Using Parallel Processing

Amey Ahire 211080006 TY(IT)
Gargi Aware 211081013 TY(IT)

June 3 2024

Abstract

This mini-project demonstrates a number plate detection system using OpenCV and parallel processing techniques. The system captures video frames, detects number plates, and processes the images in parallel to improve performance.

1 Introduction

Number plate detection is a crucial task in many applications such as traffic monitoring and automated toll collection. This project utilizes OpenCV for image processing and EasyOCR for optical character recognition (OCR). To enhance the efficiency of the system, parallel processing is employed using Python's multiprocessing module.

2 System Overview

The system captures video frames from a webcam, detects number plates in the frames, and processes the images to extract the number plates. The processing is done in parallel to ensure real-time performance.

3 Implementation

3.1 Dependencies

The following libraries are required for this project:

- `cv2` (OpenCV)
- `multiprocessing` (Python standard library)
- `easyocr`

- `matplotlib`
- `joblib`

3.2 Number Plate Detection

The number plate detection is implemented using OpenCV's Haar Cascade classifier. The video frames are captured from a webcam and processed in parallel.

Listing 1: Number Plate Detection with Parallel Processing

```
import cv2
import multiprocessing as mp

harcascade = "model/haarcascade_russian_plate_number.xml"

cap = cv2.VideoCapture(1)
cap.set(3, 640)  # width
cap.set(4, 480)  # height

min_area = 500
count = 0

def detect_plates(frame_queue, result_queue):
    plate_cascade = cv2.CascadeClassifier(harcascade)

    while True:
        img = frame_queue.get()
        if img is None:
            break

        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        plates = plate_cascade.detectMultiScale(img_gray, 1.1, 4)

        for (x, y, w, h) in plates:
            area = w * h
            if area > min_area:
                cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(img, "Number Plate", (x, y - 5), cv2.FONT_HERSHEY_CO
                img_roi = img[y: y + h, x: x + w]
                result_queue.put(img_roi)

        result_queue.put(img)

def process_frame(frame_queue):
    while True:
        success, img = cap.read()
```

```
            if not success:
                frame_queue.put(None)
                break
            frame_queue.put(img)

    def main():
        global count

        frame_queue = mp.Queue(maxsize=10)
        result_queue = mp.Queue(maxsize=10)

        # Start the plate detection process
        detection_process = mp.Process(target=detect_plates, args=(frame_queue, resu
        detection_process.start()

        # Start capturing frames
        frame_capture_process = mp.Process(target=process_frame, args=(frame_queue,)
        frame_capture_process.start()

        while True:
            img = result_queue.get()
            if img is None:
                break

            cv2.imshow("Result", img)

            if cv2.waitKey(1) & 0xFF == ord('s'):
                img_roi = result_queue.get()
                cv2.imwrite("plates/scanned_img_" + str(count) + ".jpg", img_roi)
                cv2.rectangle(img, (0, 200), (640, 300), (0, 255, 0), cv2.FILLED)
                cv2.putText(img, "Plate Saved", (150, 265), cv2.FONT_HERSHEY_COMPLEX
                cv2.imshow("Results", img)
                cv2.waitKey(500)
                count += 1

        frame_capture_process.join()
        detection_process.join()

        cap.release()
        cv2.destroyAllWindows()

    if __name__ == "__main__":
        main()
```

3.3 Screenshots

3.4 Text Recognition on Number Plates

To recognize the text on the detected number plates, EasyOCR is used. The OCR processing is also performed in parallel.

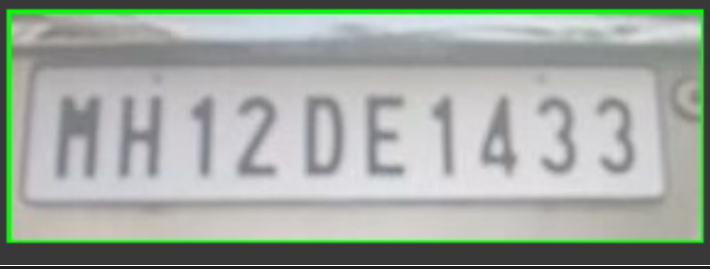
Listing 2: Text Recognition Using EasyOCR

```
!pip install easyocr
import matplotlib.pyplot as plt
import cv2
import easyocr
from IPython.display import Image
from joblib import Parallel, delayed

img_path = "scaned_img_0.jpg"

Image(img_path)

image = cv2.imread(img_path)
# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply bilateral filter to reduce noise while keeping edges sharp
filtered = cv2.bilateralFilter(gray, 11, 17, 17)

# Apply thresholding
_, thresh = cv2.threshold(filtered, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Display the preprocessed image
plt.imshow(cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB))
plt.title('Preprocessed Image')
plt.show()

reader = easyocr.Reader(['en'])

# Define a function to process OCR on a single image
def process_ocr(img):
    # Read the text from the image
    output = reader.readtext(img)
    return output

# Perform OCR in parallel
output = Parallel(n_jobs=-1)(delayed(process_ocr)(img_path) for _ in range(1))

output

cord = output[-1][0]
```

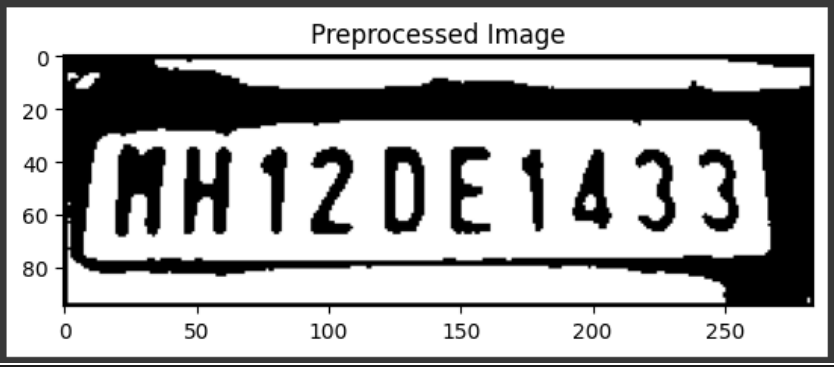
```
cord

cord, _, _ = output[-1][0]

# Find the bounding box coordinates
x_values = [coord[0] for coord in cord]
y_values = [coord[1] for coord in cord]
x_min = min(x_values)
y_min = min(y_values)
x_max = max(x_values)
y_max = max(y_values)

# Display the bounding box on the image
image = cv2.imread(img_path)
cv2.rectangle(image, (x_min, y_min), (x_max, y_max), (0, 0, 255), 2)

# Display the image with bounding box
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

3.5 Screenshots of Text Recognition

```
[ ]  Image(img_path)
```

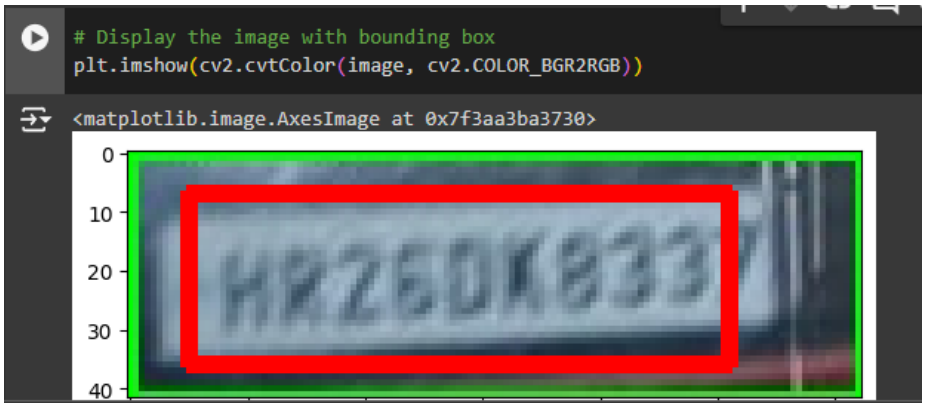



```
# Display the preprocessed image
plt.imshow(cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB))
plt.title('Preprocessed Image')
plt.show()
```




```
[ ]  output
```

```
[[([[13, 29], [262, 29], [262, 74], [13, 74]],
   '0HIZ DE1433',
   0.3642031607188293)]]
```



4 Parallel Processing Method

In this project, parallel processing is utilized to enhance the performance of number plate detection. The Python `multiprocessing` module is used to achieve this. The main idea is to separate the tasks of frame capturing and plate detection into different processes that run concurrently.

4.1 Why Parallel Processing?

In a sequential processing approach, the main program captures a frame from the video stream, processes it to detect number plates, and then displays the result. This sequence of operations can create a bottleneck, especially when real-time performance is required. By using parallel processing, we can overlap the operations of capturing frames and detecting plates, thus improving the throughput and responsiveness of the system.

4.2 Implementation Details

The implementation involves creating two separate processes:

1. **Frame Capturing Process:** Continuously captures frames from the webcam and puts them into a shared queue.
2. **Plate Detection Process:** Continuously reads frames from the shared queue, processes them to detect number plates, and puts the results into another shared queue.

Here's the code snippet showing the parallel processing setup:

Listing 3: Parallel Processing Setup

```
frame_queue = mp.Queue(maxsize=10)
result_queue = mp.Queue(maxsize=10)
```

```
# Start the plate detection process
detection_process = mp.Process(target=detect_plates, args=(frame_queue, result_qu
detection_process.start()

# Start capturing frames
frame_capture_process = mp.Process(target=process_frame, args=(frame_queue,))
frame_capture_process.start()

# Main loop to display results
while True:
    img = result_queue.get()
    if img is None:
        break

    cv2.imshow("Result", img)

    if cv2.waitKey(1) & 0xFF == ord('s'):
        img_roi = result_queue.get()
        cv2.imwrite("plates/scanned_img_" + str(count) + ".jpg", img_roi)
        cv2.rectangle(img, (0, 200), (640, 300), (0, 255, 0), cv2.FILLED)
        cv2.putText(img, "Plate Saved", (150, 265), cv2.FONT_HERSHEY_COMPLEX_SMA
        cv2.imshow("Results", img)
        cv2.waitKey(500)
        count += 1

frame_capture_process.join()
detection_process.join()

cap.release()
cv2.destroyAllWindows()
```

5 Comparison with Non-Parallel Method

5.1 Non-Parallel Method

In the non-parallel method, all operations (frame capturing, plate detection, and result display) are performed sequentially in a single process. Here is a simplified version of how the code would look without parallel processing:

Listing 4: Non-Parallel Method

```
cap = cv2.VideoCapture(1)
cap.set(3, 640)
cap.set(4, 480)
min_area = 500
count = 0
```

```
plate_cascade = cv2.CascadeClassifier("model/haarcascade_russian_plate_number.xm

while True:
    success, img = cap.read()
    if not success:
        break

    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plates = plate_cascade.detectMultiScale(img_gray, 1.1, 4)

    for (x, y, w, h) in plates:
        area = w * h
        if area > min_area:
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(img, "Number Plate", (x, y - 5), cv2.FONT_HERSHEY_COMPLE
            img_roi = img[y: y + h, x: x + w]
            cv2.imwrite("plates/scanned_img_" + str(count) + ".jpg", img_roi)
            count += 1

    cv2.imshow("Result", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

5.2 Performance Comparison

- **Sequential Processing:** In the non-parallel approach, the frame rate can be significantly lower because each frame has to go through the complete process of capturing, detecting, and displaying sequentially. This can lead to delays and reduced real-time performance.
- **Parallel Processing:** By separating the frame capturing and plate detection into different processes, the system can handle more frames per second, leading to smoother and faster real-time detection. The overlapping of operations allows the system to utilize CPU resources more efficiently, reducing the idle time between operations.

5.3 Advantages of Parallel Processing

- **Improved Throughput:** More frames can be processed per second.
- **Reduced Latency:** Faster response time for real-time applications.
- **Better Resource Utilization:** Efficient use of multi-core processors.

5.4 Disadvantages of Parallel Processing

- **Complexity:** More complex code structure and debugging.
- **Inter-Process Communication Overhead:** Potential overhead due to data exchange between processes.

6 Conclusion

This project demonstrates the implementation of a number plate detection system using parallel processing to improve efficiency. By leveraging OpenCV for image processing and EasyOCR for text recognition, the system can effectively detect and recognize number plates in real-time. The parallel processing approach significantly enhances performance compared to a sequential method, making it suitable for applications requiring high throughput and low latency.