Assignment 2

SOFE 4590U Embedded Systems

Hanzalah Patel - 100785574

GitHub Link: https://github.com/Hxnzo/CarDetection

ONTARIO TECH UNIVERSITY
FACULTY OF ENGINEERING AND APPLIED SCIENCE

Due Date: November 21, 2024

Architecture 1: Debian x86_64

Setup:

1. cd into the folder where QEMU is downloaded along with the img files:
    a. **cd C:\Program Files\qemu**
2. Start the x86_64 virtual machine using QEMU:
    a. **qemu-system-x86_64 -m 8G -drive file=debian_disk_x86_64.img,format=qcow2**
3. Install necessary dependencies:
    a. **sudo apt-get update**
    b. **sudo apt-get install git python3 python3-pip**
4. Clone the Git repository containing the object detection program:
    a. **git clone https://github.com/Hxnzo/CarDetection.git**
    b. **cd CarDetection**
5. Set up a virtual environment:
    a. **sudo apt install virtualenv**
    b. **virtualenv venv**
    c. **source venv/bin/activate**
6. Install OpenCV (for Python):
    a. **pip install opencv-python**

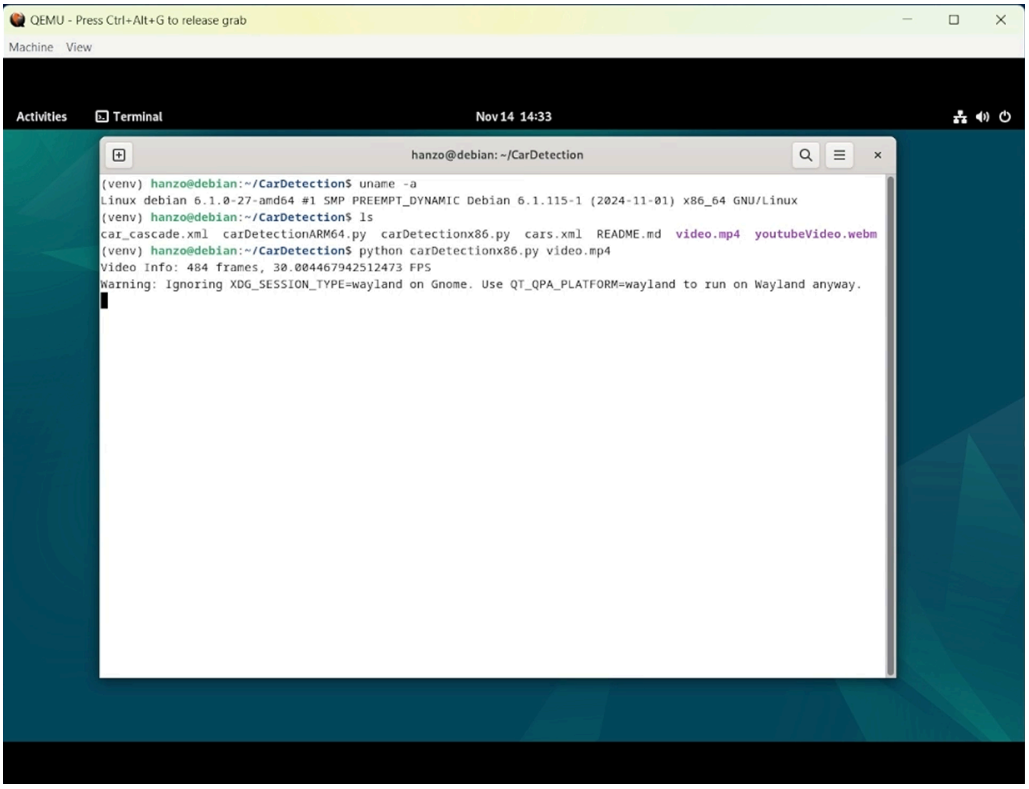
Run Object Detection Code:

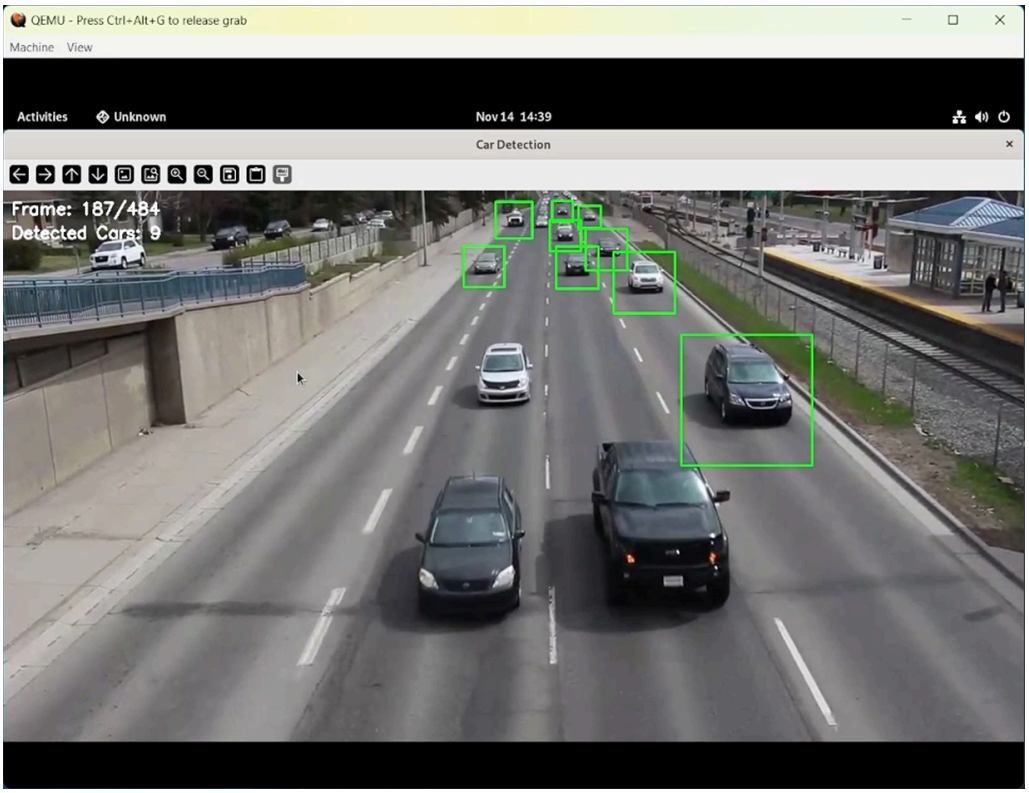
1. Execute the object detection program on a video (e.g., video.mp4):
    a. **python3 carDetectionx86.py video.mp4**

Video of demonstration for x86_64:

Link: https://youtu.be/IyLe23DC3JI

Snapshots for x86_64 Architecture Running Successfully:





Architecture 2: Debian ARM64

Setup:

1. cd into the folder where QEMU is downloaded along with the img files:
    a. **cd C:\Program Files\qemu**
2. Run the command to boot and launch the architecture:
    a. **qemu-system-aarch64 -m 8192 -M virt -cpu cortex-a57 -bios "QEMU_EFI.fd" -drive file=debian_disk_arm64.img,format=qcow2,if=none,id=hd -device virtio-blk-device,drive=hd -device virtio-net-device,netdev=net0 -netdev user,id=net0 -serial mon:stdio**
3. Once the system is booted you will see a shell terminal. Enter these commands in order:
    a. **FS0:**
    b. **cd EFI**
    c. **cd Debian**
    d. **grubaa64.EFI**
4. Install necessary dependencies:
    a. **sudo apt-get update**
    b. **sudo apt-get install git python3 python3-pip virtualenv**
5. Clone the Git repository containing the object detection program:
    a. **git clone https://github.com/Hxnzo/CarDetection.git**
    b. **cd CarDetection**
6. Set up a virtual environment:
    a. **virtualenv venv**
    b. **source venv/bin/activate**
7. Install OpenCV (headless version) for Python:
    a. **pip install opencv-python-headless**

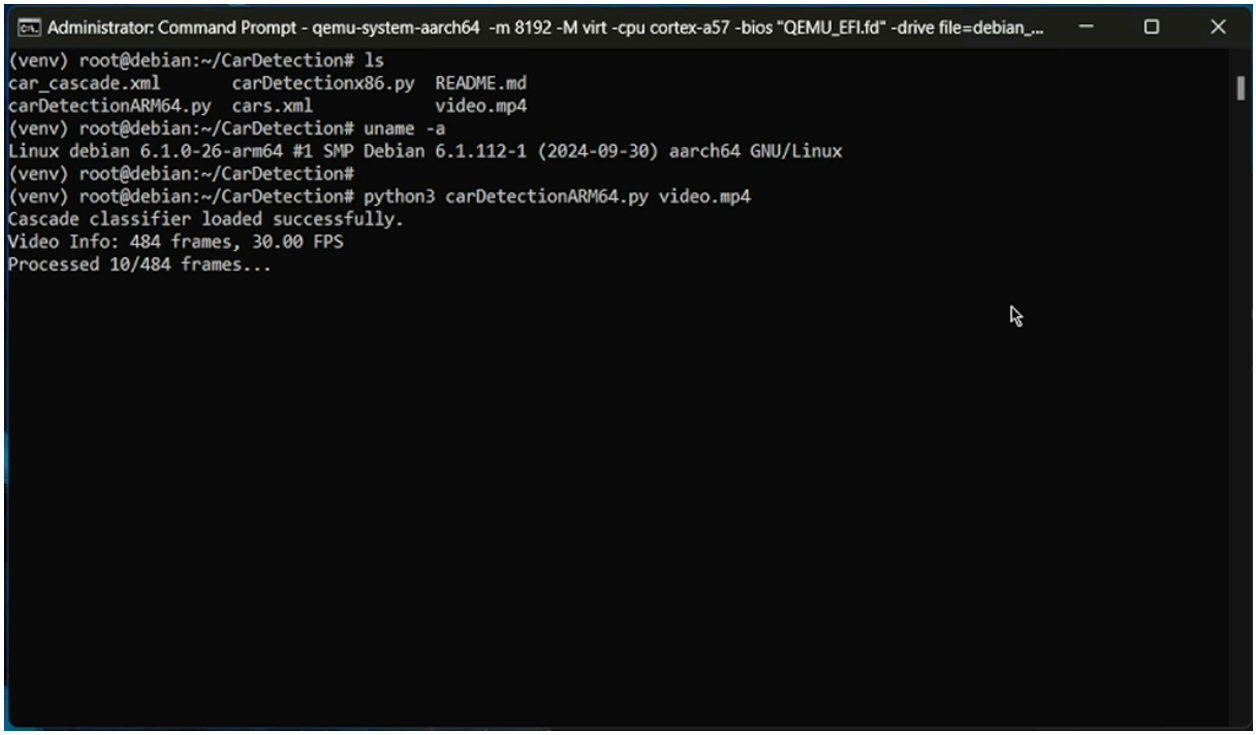
Run Object Detection Code:

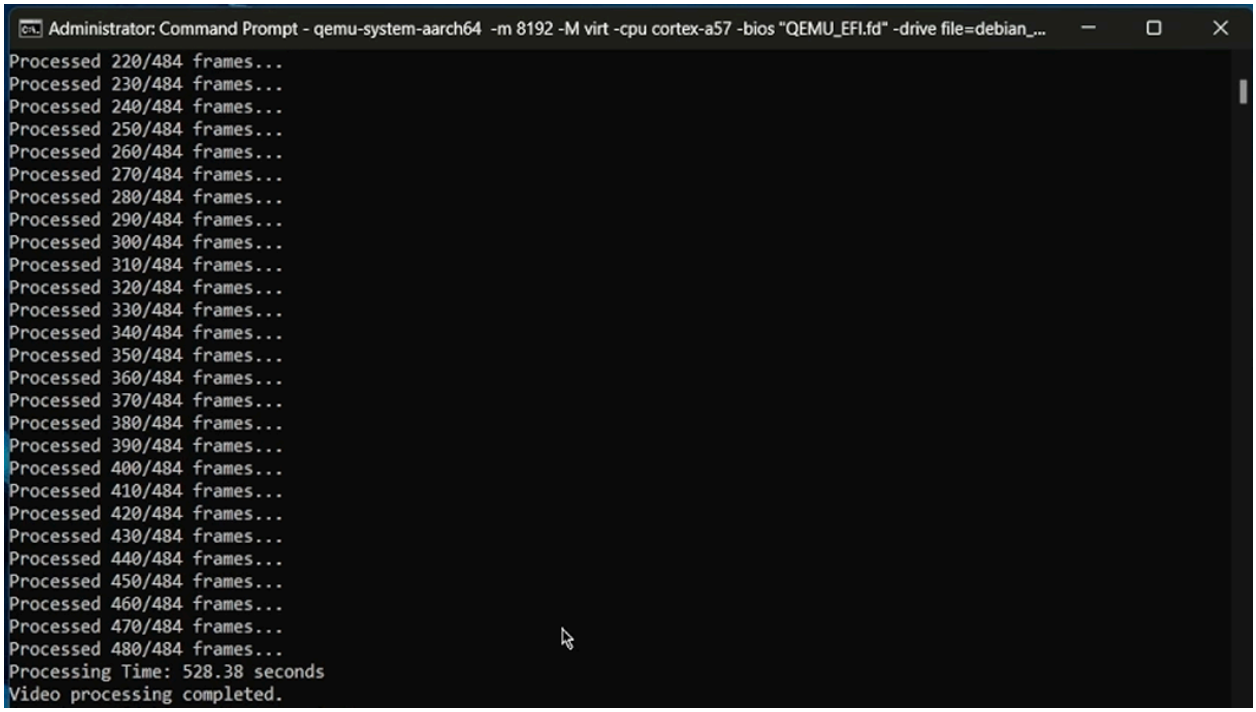
1. **Execute the object detection program on a video (e.g., video.mp4):**
    a. **python3 carDetectionARM64.py video.mp4**

Video of demonstration for ARM64:

Link: https://youtu.be/Pzh-m6Drg8g

Snapshots for x86_64 Architecture Running Successfully:




Administrator: Command Prompt - qemu-system-aarch64 -m 8192 -M virt -cpu cortex-a57 -bios "QEMU_EFI.fd" -drive file=debian_...
Processed 220/484 frames...
Processed 230/484 frames...
Processed 240/484 frames...
Processed 250/484 frames...
Processed 260/484 frames...
Processed 270/484 frames...
Processed 280/484 frames...
Processed 290/484 frames...
Processed 300/484 frames...
Processed 310/484 frames...
Processed 320/484 frames...
Processed 330/484 frames...
Processed 340/484 frames...
Processed 350/484 frames...
Processed 360/484 frames...
Processed 370/484 frames...
Processed 380/484 frames...
Processed 390/484 frames...
Processed 400/484 frames...
Processed 410/484 frames...
Processed 420/484 frames...
Processed 430/484 frames...
Processed 440/484 frames...
Processed 450/484 frames...
Processed 460/484 frames...
Processed 470/484 frames...
Processed 480/484 frames...
Processing Time: 528.38 seconds
Video processing completed.

How to downloaded the youtube video provided on both x86_64 and ARM64 architectures:

1. Download yt-dlp package using:
    a. **pip install -U yt-dlp**
2. Use yt-dlp to download the video:
    a. **yt-dlp -f 313 MNn9qKG2UFI**

****NOTE****

I did not use the youtube video provided as it was very large (9840 frames and 4k video which made it especially large) and It would have taken the detection algorithm too long to get through. I did download it using the instructions above and it worked fine but It just took a long time to run the algorithm. Which is why I used another video.

The video I used is linked in my GitHub repository (linked on first page).

Analysis and Conclusion:

The following analysis will speak towards the time taken in seconds to execute the object detection algorithm on x86_64 and ARM64 architectures using the video.mp4 in the GitHub repository. The differences in execution time shows the performance variations observed while the code was running. The x86_64 architecture took 1006 seconds to complete the algorithm on the video whereas the ARM64 architecture took 528 seconds.

The processing time difference between x86_64 and ARM64 architectures in this car detection algorithm likely comes from both architectural differences and variations in how the code handles output and frame processing. For example, the x86_64 code includes frequent on-screen display updates and prints the processing time for each frame, which can slow down the processing as it adds extra work for the CPU. In contrast, the ARM64 code reduces the frequency of console output by only logging every 10 frames, which can help it process more frames faster.

The algorithm is also resource-intensive because it processes every video frame individually, converting it to grayscale, detecting cars, and drawing rectangles. Each of these tasks takes time, especially when run sequentially. Additionally, factors like limited hardware acceleration on either architecture, or differences in how they handle OpenCV’s library functions, may also contribute to the total processing time.

In conclusion, the car detection algorithm's execution time varied significantly between x86_64 and ARM64 architectures, with x86_64 taking longer due to frequent output updates and frame processing overhead. ARM64 achieved faster results by reducing output frequency, allowing more efficient frame handling. Both versions faced delays from the resource-intensive process of converting each frame, detecting cars, and drawing shapes. These results highlight the impact of architectural differences and code optimizations on performance in video analysis tasks

References:

https://github.com/TegSingh/QEMU-object-detection