GH repo: https://github.com/evgerritz/cpsc429_labs/tree/2_p3
Steps to run:
On Zoo:

1. Boot server vm and forward the VNC and server ports

On Host:

1. Use VNC to connect to server
2. Enter movenet_server/
3. cargo run

On Target:

5. Connect camera to VM
6. Start yuv422 webcam with ./fake_webcam
7. Enter rust_movenet/
8. ./tunnel (to connect to the Zoo)
9. cargo run

Part 3:

- Acknowledgements
    - Resources provided in assignment
    - Mmap streaming example: https://www.kernel.org/doc/html/v4.9/media/uapi/v4l/mmap.html
    - Videodev2 header: https://www.kernel.org/doc/html/v5.7/media/uapi/v4l/videodev.html
    - Another v4l example: https://medium.com/@athul929/capture-an-image-using-v4l2-api-5b6022d79e1d
    - Setting up virtual yuv422 webcam: https://stackoverflow.com/questions/59574987/how-to-change-mjpeg-to-yuyv422-from-a-webcam-to-a-v4l2loopback
    - Code I translated into rust for converting from yuv422 to rgb: https://github.com/kd40629rtlrtl/yuv422_to_rgb/blob/master/yuv_to_rgb.c
- Challenges
    - Getting the virtual webcam set up
    - Creating Rust structs that were compatible with the ioctl calls was difficult. I had to use gdb to print the physical memory layout of the v4l2_buffer struct in order to get the alignment right.
    - Debugging failed ioctl calls
    - Converting between libc::c_void and &[u8]

    - Reading a stream of RGB bytes into an OpenCV Mat
    - Debugging when images were incorrectly converted
- Overview
    - The majority of the code for this assignment is in rust_movenet/v4l_utils.rs.
        - v4l_utils.rs implements the Rust interface for v4l, as will be used by the client to capture images from the webcam.
        - main.rs has been rewritten to use the v4l wrappers in v4l_utils.rs.
    - movenet_server/main.rs now includes a yuv422-to-rgb24 conversion function, and the resize_with_padding function from rust_movenet/util.rs.
    - The general workflow is as follows:
        - The client uses v4l to capture a yuv422 buffer, and sends it as bytes to the server.
        - The server then converts that yuv422 buffer to rgb24, creates an OpenCV matrix, and then does the required resizing of input for the network.
        - The server then runs the DNN interpreter, and sends the results as bytes back to the client.
        - The client converts these bytes to floats, draws the output on a blank frame, and then displays the frame.