GH repo: https://github.com/evgerritz/cpsc429_labs/tree/2_p2
Steps to run:
On Zoo:

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

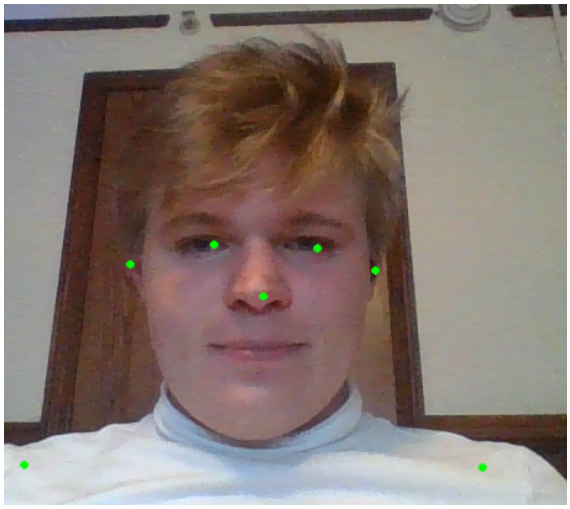
On Host:

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

On Target:

1. Connect camera to VM
2. Enter rust_movenet/
3. ./tunnel
4. cargo run

Part 1:

- Acknowledgments
  - Used the provided resources
- Challenges
  - None
- 

Part 2:

- Acknowledgments
    - Used the provided resources
    - https://riptutorial.com/rust/example/4404/a-simple-tcp-client-and-server-application--echo
- Challenges
    - Getting all of the forwarding to work in order to connect to the VM on the Zoo was tricky.
    - Ultimately, I could not get kvm acceleration to work on the zoo (the zoo uses qemu version 2.3), which made it impossible to run the rust movenet server (tflitec had not finished building after many hours). I was however, able to get a basic server connected and successfully sent and received data from that. The rest of the assignment, however, was done using a server and client on the same virtual machine.
    - Coordinating the reads and writes between the server and the client was also difficult until I switched to using a BufReader. This allowed me to use the read_exact method, which gave correct results from the server.
- Overview/Discussion
    - I chose to divide the server and the client's tasks by having the client send and receive as bytes the input and output, respectively, of the network running on the server. 'client.rs' contains the code for connecting to and sending bytes to/from the server, as well as some functions for converting to/from bytes. The movenet_server cargo project contains the code for the server-side DNN interpreter. It accepts TCP streams and then repeatedly reads input from the TCP stream, feeds that into the network, and then writes the output of the network back to the TCP stream.
    - For correctness, I ensured that the resulting image matched the original results when. I also used read_exact to guarantee that each frame would be read individually and at once by the server. Initially, my results were incorrect because of incomplete reads.
    - For performance, I calculated the number of images the system was capable of processing per second and got an average value of 7.6 images processed per second.
    - For the app interface, I simplified interactions with the network by creating a Server struct that contains a TcpStream. The client can send or receive bytes to the server through corresponding methods. I also wrote some functions to simplify the client's conversion of data to and from bytes. From the client's perspective, all it needs to do is convert the data to bytes, send the data to the

server, receive and convert the output from the server, and display the result. To ensure good networking performance, I decided to send each frame at once to the server instead of splitting it into smaller chunks. Since the data can always fit into two TCP packets, it is not likely not too big for chunks of this size to be an issue.