GH repo: https://github.com/evgerritz/cpsc429_labs/tree/3_p1
Steps to run:
On Zoo:

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

On Host:

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

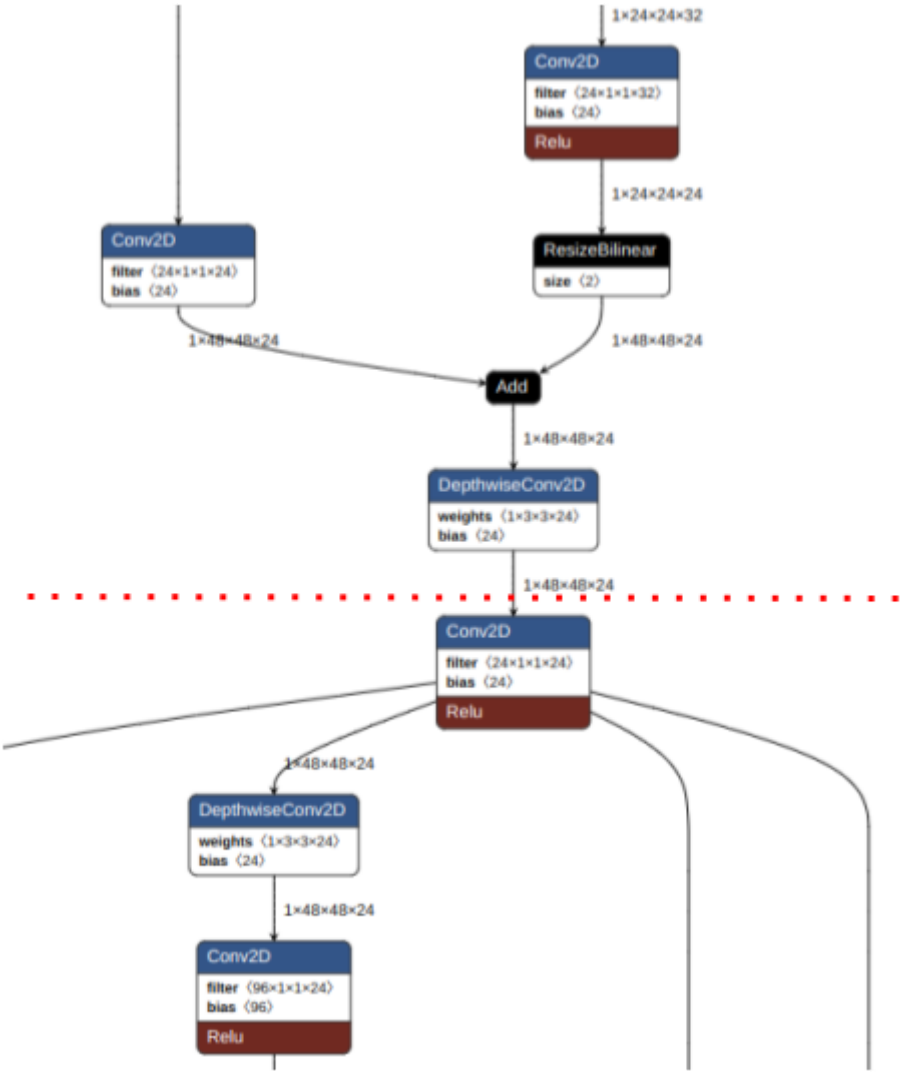
On Target:

1. Connect camera to VM
2. Start yuv422 webcam with ./fake_webcam
3. Enter rust_movenet/
4. ./tunnel (to connect to the Zoo)
5. cargo run

Part 1:

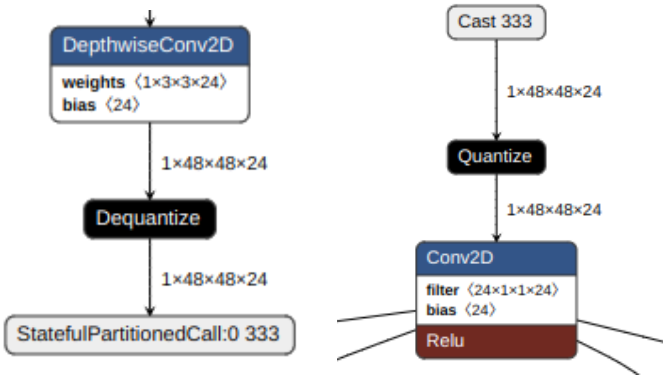
- Acknowledgements
    - Resources provided in the assignment
    - Example of using FlatBufferBuilder in Python: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/tools/flatbuffer_utils.py
    - https://google.github.io/flatbuffers/flatbuffers_guide_use_rust.html
- Challenges
    - Figuring out how to use the schema_generated.rs bindings was tricky as there was a lot of code with no documentation. Guojun was very helpful in office hours in pointing me toward the right functions for reading in the model. After that, it was just a matter of learning how to use the various fields of the ModelT. Figuring out how to save the model was also unclear, but a python file (see acknowledgments above) I found online was useful.
    - The other big challenge was figuring out how to add the Quantize/Dequantize operators. I ended up printing all of the tensors, buffers, and operators in the original network into a file (all_info) and then spent a lot of time reading through that to make sense of how all of the data structures fit together. Netron.app was also invaluable in making sure the tensors were correctly connected and that the input/outputs of the network itself were correct.
- Overview

- The majority of the code for this assignment is in splitter/, which uses Rust to split the tflite neural network in half and produces splitter/output/upper.tflite and splitter/output/lower.tflite.
  - The splitter first reads in the original tflite file and uses it to create two ModelTs, one for the upper network and one for the lower network.
  - I decided to split the network at tensor 255 (red line in the screenshot below), as this was the smallest tensor that had at least one Conv2D operation applied after it and did not require me to send multiple tensors across the network (i.e. because the network's processing had multiple branches). Splitting at the smallest possible tensor was theoretically useful because sending as few bytes as possible over the network will make the program faster.



  - I then looped through the model's operators to find the index of the operator that outputs to 255. I used this index to split the operators between the upper and lower halves by calling split_offset.

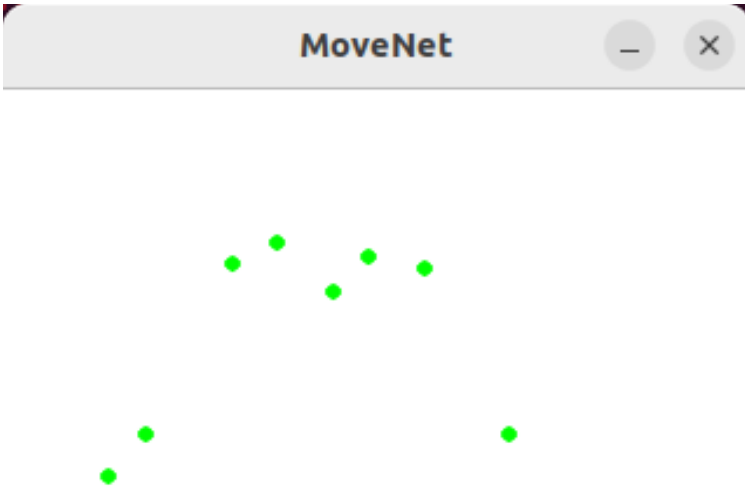
- Because the Rust tflitec crate doesn't support the type of the intermediate output of the network (Int8), I had to add a Dequantize operator to the end of the upper network to convert to float32, as well as a corresponding Quantize operator at the beginning of the lower network to convert back to Int8.
    - To do this, we must first create two new tensors for the new operators' input (in the lower model) / output (in the upper model)



    - The new tensors are created by creating a new empty buffer, adding it to the model's buffers, and then creating a new TensorT with the appropriate shape that uses the empty buffer. We then need to add these tensors to the model's list of tensors
    - Now, we can actually create the operators. I copied the opcode_index from the original network's Quantize and Dequantize operators and then set input/output appropriately using our new tensor (333) and the tensor we split the network at (255).
    - Finally, we need to add these operators to the network
- Next we need to change the output of the upper network and the input of the lower network to our new tensor (333)
- The two models have now been correctly split so we need to build them using a new FlatBufferBuilder by calling pack and then finish. We can then call finished_data on the builder to get the bytes of the model's flatbuffer representation, which are then written to our two output files.

- rust_movenet was updated to start an interpreter for the upper model and sends the output of that network to the server.
- movenet_server was updated to start the lower model, and sends its output back to the user program, which displays the output.

- Here is a sample output:



  - The model appears to detect my nose, eyes, and shoulders, as it has in the previous assignments.