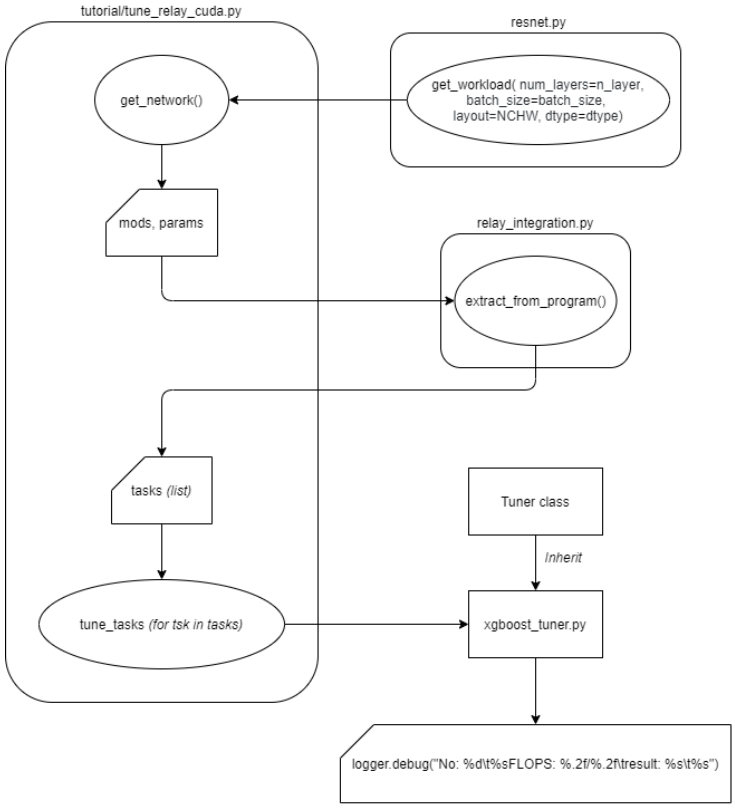
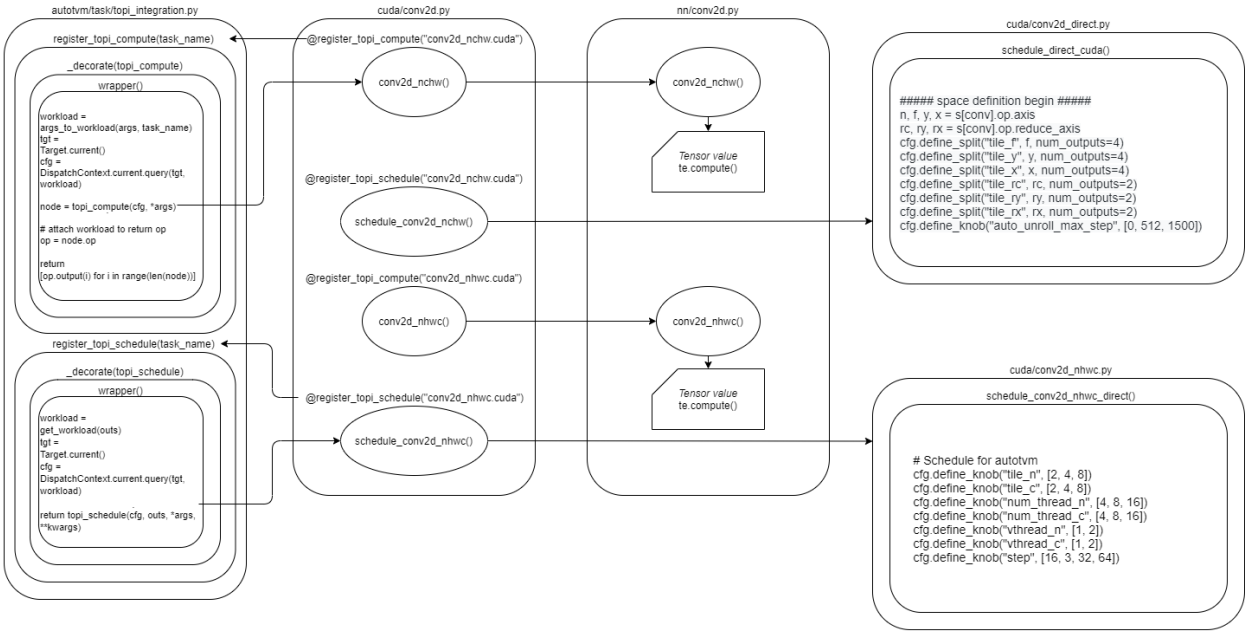
CHOI Jang Hyeon (1155077214)
Supervisors: Professors JIA Jiaya and YU Bei
Mentor: Mr. QI Sun
CSE Department
Term 2, 2020-21
March 29th, 2021

FYP Progress Report: Week 12

To recap, the goals of this research project is to: 1. automate the memory layout optimization, and 2. Accelerate the configuration exploration process. The memory layout can be automatically optimized by combining the configurations for both NCHW and NHWC into one and then tune them together. To achieve this, I have investigated which variables should be dealt with in what function or Python file; the module and the parameters returned by resnet cannot be combined before extracting tasks according to them because the module is an IR Function class, so I had to proceed to lower-levels of the code.





With advice from my mentor, concatenating the tensor values for both layouts would be the most appropriate approach, which will ultimately enlarge the configuration space.

I have also thought of accelerating the configuration space by providing different tiling factors specific to the hardware, but the code is so hard-coded that additional modifications needs to be made. For instance, if I increased the number of axes to 8 in tile_f, then additional thread axes need to be added as well other than bf, vf, tf, and fi:

```
cfg.define_split("tile_f", f, num_outputs=4)
cfg.define_split("tile_y", y, num_outputs=4)
cfg.define_split("tile_x", x, num_outputs=4)
cfg.define_split("tile_rc", rc, num_outputs=2)
cfg.define_split("tile_ry", ry, num_outputs=2)
cfg.define_split("tile_rx", rx, num_outputs=2)
cfg.define_knob("auto_unroll_max_step", [0, 512, 1500])

# tile and bind spatial axes
n, f, y, x = s[output].op.axis
kernel_scope, n = s[output].split(n, nparts=1)

bf, vf, tf, fi = cfg["tile_f"].apply(s, output, f)
by, vy, ty, yi = cfg["tile_y"].apply(s, output, y)
bx, vx, tx, xi = cfg["tile_x"].apply(s, output, x)

bf = s[output].fuse(n, bf)
s[output].bind(bf, te.thread_axis("blockIdx.z"))
s[output].bind(by, te.thread_axis("blockIdx.y"))
s[output].bind(bx, te.thread_axis("blockIdx.x"))
s[output].bind(vf, te.thread_axis("vthread"))
s[output].bind(vy, te.thread_axis("vthread"))
s[output].bind(vx, te.thread_axis("vthread"))
s[output].bind(tf, te.thread_axis("threadIdx.z"))
s[output].bind(ty, te.thread_axis("threadIdx.y"))
s[output].bind(tx, te.thread_axis("threadIdx.x"))
s[output].reorder(bf, by, bx, vf, vy, vx, tf, ty, tx, fi, yi, xi)
s[OL].compute_at(s[output], tx)
```