**PyMetaController Algorithm**

import torch.multiprocessing as mp

if \_\_name\_\_ == ‘\_\_main\_\_’:

n\_train\_processes = 4

shared\_model = **ActorCritic()**

shared\_model.share\_memory()

processes = []

for rank in range(n\_train\_processes + 1):

if rank == 0:

p = mp.Process(target = **test**, args = (shared\_model,))

else:

p = mp.Process(target = **train**, args = (shared\_model, rank, ))

p.start()

processes.append(p)

for p in processes:

p.join()

**ActorCritic class design**

import torch

import torch.nn as nn

import torch.nn.functional as F

class ActorCritic(nn.Module):

def \_\_init\_\_(self):

super(ActorCritic, self).\_\_init\_\_()

# suppose input width, height is W, H respectively.

# OutputHeight for CNN = (H + 2P – FH)/S + 1

self.conv1 = nn.Conv2d(3, 32, 7, stride=2, padding=1)

self.conv2 = nn.Conv2d(32, 32, 5, stride=1, padding=1)

self.conv3 = nn.Conv2d(32, 32, 3, stride=1, padding=1)

# Spatial summation ??? fc → gru OR gru → fc

self.fc1 = nn.Linear(\_\_, 1)

self.gru = nn.GRU(\_\_, hidden\_size, 1)

def forward(self, input):

x = F.elu(self.conv1(inputs))

x = F.elu(self.conv2(inputs))

x = F.elu(self.conv3(x))

x = self.fc1(x)

x = self.gru(x)

prob = F.softmax(x, dim=1)

return prob