# Supplementary Material

## 1. Structure of the policy network

We use a VGG-16 style architecture as our policy network. Different from the vanilla VGG-16, which is designed for image classification, we use 1D convolution instead. The detailed architecture of the policy network is presented in Table 1.

Table 1: The network architecture of the policy network. $N$ is the number of filters in the CNN to be pruned.

| Index | Layer | Type | Feature map | Kernel size | Stride | Output size | Activation |
|---|---|---|---|---|---|---|---|
| 0 | Input | Feature | 1 | - | - | Nx7 | - |
| 1 | conv1_1 | 1D conv | 64 | 3x3 | 1 | Nx64 | ReLu |
| 2 | conv1_2 | 1D conv | 64 | 3x3 | 1 | Nx64 | ReLu |
| 3 | pool1 | Pool | - | 2x2 | 2 | (N/2)x64 | Max |
| 4 | conv2_1 | 1D conv | 128 | 3x3 | 1 | (N/2)x128 | ReLu |
| 5 | conv2_2 | 1D conv | 128 | 3x3 | 1 | (N/2)x128 | ReLu |
| 6 | pool2 | Pool | - | 2x2 | 2 | (N/4)x128 | Max |
| 7 | conv3_1 | 1D conv | 256 | 3x3 | 1 | (N/4)x256 | ReLu |
| 8 | conv3_2 | 1D conv | 256 | 3x3 | 1 | (N/4)x256 | ReLu |
| 8 | conv3_3 | 1D conv | 256 | 3x3 | 1 | (N/4)x256 | ReLu |
| 10 | pool3 | Pool | - | 2x2 | 2 | (N/8)x256 | Max |
| 11 | conv4_1 | 1D conv | 512 | 3x3 | 1 | (N/8)x512 | ReLu |
| 12 | conv4_2 | 1D conv | 512 | 3x3 | 1 | (N/8)x512 | ReLu |
| 13 | conv4_3 | 1D conv | 512 | 3x3 | 1 | (N/8)x512 | ReLu |
| 14 | pool4 | Pool | - | 2x2 | 2 | (N/16)x512 | Max |
| 15 | conv5_1 | 1D conv | 512 | 3x3 | 1 | (N/16)x512 | ReLu |
| 16 | conv5_2 | 1D conv | 512 | 3x3 | 1 | (N/16)x512 | ReLu |
| 17 | conv5_3 | 1D conv | 512 | 3x3 | 1 | (N/16)x512 | ReLu |
| 18 | pool5 | Pool | - | 2x2 | 2 | (N/32)x512 | Max |
| 19 | fc6_1 | fc ($\pi$) | - | - | - | N | Softmax |
| 19 | fc6_2 | fc ($v$) | - | - | - | 1 | Tanh |

## 2. Pseudocodes of key components in our approach.

**Algorithm 1** Get the improved policy $\boldsymbol{\pi}$ after MCTS search: getPolicyPi($s_i$)

**Input:** Current configuration of the network to be pruned $s_i$, number of MCTS simulations per action $n_{mcts}$, total number of filters in the network to be pruned $n_f$, temperature $\tau$.
**Output:** $\boldsymbol{\pi}_i$

**for** $i$ in range($n_{mcts}$) **do**
    MCTS($s_i$)
Get $N(s_i, a)$ after MCTS simulations.
**if** $\tau = 0$ **then**
    bestAction = $\mathrm{argmax}_a \, N(s_i, a)$
    $\boldsymbol{\pi}[\text{bestAction}] = 1$
**else**
    $\boldsymbol{\pi} = \frac{N(s_i,a)^{(1/\tau)}}{\sum_b N(s_i,b)^{(1/\tau)}}$
**return** $\boldsymbol{\pi}$

**Algorithm 2** Get training samples from a single iteration: getTrainSamples($s_0$)

**Input:** The raw network to be pruned $s_0$, pruning ratio $\gamma$, trainingAccBaseline b,
**Output:** trainSamples $(s_i, \boldsymbol{\pi}_i, v)$

$t = 0$, $s_t = s_0$
trainSamples = []
**while** FLOPs$(s_t)/$FLOPs$(s_0) > \gamma$ **do**
    $\boldsymbol{\pi}_t$ = getPolicyPi($s_t$)
    trainSamples.append([$s_t, \boldsymbol{\pi}_t$])
    nextAction = randomChoice($\boldsymbol{\pi}_t$)
    $s_{t+1}$ = pruneFilter($s_t$,nextAction)
    $t = t + 1$
**if** trainAcc$(s_t) > b$ **then**
    $v = 1$
**else**
    $v = -1$
trainSamples = [$(x[0], x[1], v)$ **for** $x$ in trainSamples]
**return** trainSamples

**Algorithm 3** Learn to get the slimmed CNN with RL and MCTS

**Input:** The raw network to be pruned $s_0$, neural network for pruning action selection $f_\theta$, number of self-play simulations $n_{sim}$, maximum training queue length $L$.
**Output:** The optimal slimmed CNN $s_p$

totalTrainingQueue = []
**while** stopCounter $< n_s$ **do**
  **for** $i$ in range($n_{sim}$) **do**
    Initialize MCTS
    trainingSamples = getTrainSamples($s_0$)
    **if** len(totalTrainingQueue) $> L$ **then**
      totalTrainingQueue.pop()
    totalTrainingQueue += trainingSamples
  $f_\theta =$ RLTrain(totalTrainingQueue, $f_\theta$)
  Get the slimmed network $s'$ by pruning $s_0$ with $f_\theta$
  **if** trainAcc$(s') > b$ **then**
    $b =$ trainAcc$(s')$
    $s_p = s'$
    stopCounter = 0
  **else**
    stopCounter += 1
**return** $s_p$