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.

<table>
<thead>
<tr>
<th>Index</th>
<th>Layer</th>
<th>Type</th>
<th>Feature map</th>
<th>Kernel size</th>
<th>Stride</th>
<th>Output size</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input</td>
<td>Feature</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>Nx7</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>conv1_1</td>
<td>1D conv</td>
<td>64</td>
<td>3x3</td>
<td>1</td>
<td>Nx64</td>
<td>ReLu</td>
</tr>
<tr>
<td>2</td>
<td>conv1_2</td>
<td>1D conv</td>
<td>64</td>
<td>3x3</td>
<td>1</td>
<td>Nx64</td>
<td>ReLu</td>
</tr>
<tr>
<td>3</td>
<td>pool1</td>
<td>Pool</td>
<td>-</td>
<td>2x2</td>
<td>2</td>
<td>(N/2)x64</td>
<td>Max</td>
</tr>
<tr>
<td>4</td>
<td>conv2_1</td>
<td>1D conv</td>
<td>128</td>
<td>3x3</td>
<td>1</td>
<td>(N/2)x128</td>
<td>ReLu</td>
</tr>
<tr>
<td>5</td>
<td>conv2_2</td>
<td>1D conv</td>
<td>128</td>
<td>3x3</td>
<td>1</td>
<td>(N/2)x128</td>
<td>ReLu</td>
</tr>
<tr>
<td>6</td>
<td>pool2</td>
<td>Pool</td>
<td>-</td>
<td>2x2</td>
<td>2</td>
<td>(N/4)x128</td>
<td>Max</td>
</tr>
<tr>
<td>7</td>
<td>conv3_1</td>
<td>1D conv</td>
<td>256</td>
<td>3x3</td>
<td>1</td>
<td>(N/4)x256</td>
<td>ReLu</td>
</tr>
<tr>
<td>8</td>
<td>conv3_2</td>
<td>1D conv</td>
<td>256</td>
<td>3x3</td>
<td>1</td>
<td>(N/4)x256</td>
<td>ReLu</td>
</tr>
<tr>
<td>9</td>
<td>conv3_3</td>
<td>1D conv</td>
<td>256</td>
<td>3x3</td>
<td>1</td>
<td>(N/4)x256</td>
<td>ReLu</td>
</tr>
<tr>
<td>10</td>
<td>pool3</td>
<td>Pool</td>
<td>-</td>
<td>2x2</td>
<td>2</td>
<td>(N/8)x256</td>
<td>Max</td>
</tr>
<tr>
<td>11</td>
<td>conv4_1</td>
<td>1D conv</td>
<td>512</td>
<td>3x3</td>
<td>1</td>
<td>(N/8)x512</td>
<td>ReLu</td>
</tr>
<tr>
<td>12</td>
<td>conv4_2</td>
<td>1D conv</td>
<td>512</td>
<td>3x3</td>
<td>1</td>
<td>(N/8)x512</td>
<td>ReLu</td>
</tr>
<tr>
<td>13</td>
<td>conv4_3</td>
<td>1D conv</td>
<td>512</td>
<td>3x3</td>
<td>1</td>
<td>(N/8)x512</td>
<td>ReLu</td>
</tr>
<tr>
<td>14</td>
<td>pool4</td>
<td>Pool</td>
<td>-</td>
<td>2x2</td>
<td>2</td>
<td>(N/16)x512</td>
<td>Max</td>
</tr>
<tr>
<td>15</td>
<td>conv5_1</td>
<td>1D conv</td>
<td>512</td>
<td>3x3</td>
<td>1</td>
<td>(N/16)x512</td>
<td>ReLu</td>
</tr>
<tr>
<td>16</td>
<td>conv5_2</td>
<td>1D conv</td>
<td>512</td>
<td>3x3</td>
<td>1</td>
<td>(N/16)x512</td>
<td>ReLu</td>
</tr>
<tr>
<td>17</td>
<td>conv5_3</td>
<td>1D conv</td>
<td>512</td>
<td>3x3</td>
<td>1</td>
<td>(N/16)x512</td>
<td>ReLu</td>
</tr>
<tr>
<td>18</td>
<td>pool5</td>
<td>Pool</td>
<td>-</td>
<td>2x2</td>
<td>2</td>
<td>(N/32)x512</td>
<td>Max</td>
</tr>
<tr>
<td>19</td>
<td>fc6_1</td>
<td>fc (( \pi ))</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>N</td>
<td>Softmax</td>
</tr>
<tr>
<td>19</td>
<td>fc6_2</td>
<td>fc (( v ))</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>Tanh</td>
</tr>
</tbody>
</table>
2. Pseudocodes of key components in our approach.

Algorithm 1 Get the improved policy $\pi$ 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:** $\pi_i$

1: `for i in range($n_{mcts}$) do`
2: `MCTS($s_i$)`
3: `Get $N(s_i,a)$ after MCTS simulations.`
4: `if $\tau = 0$ then`
5: `bestAction = argmax_a $N(s_i,a)$`
6: `$\pi[\text{bestAction}] = 1$`
7: `else`
8: `$\pi = \frac{N(s_i,a)^{1/\tau}}{\sum_b N(s_i,b)^{1/\tau}}$`

`return $\pi$`

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 ($s_i$, $\pi_i$, $v$)

1: `t = 0, $s_t = s_0$`
2: `trainSamples = []`
3: `while FLOPs($s_t$)/FLOPs($s_0$) > $\gamma$ do`
4: `$\pi_t = \text{getPolicyPi}(s_t)$`
5: `trainSamples.append([s_t, $\pi_t$])`
6: `nextAction = randomChoice($\pi_t$)`
7: `$s_{t+1} = \text{pruneFilter}(s_t, \text{nextAction})$`
8: `t = t + 1`
9: `if trainAcc($s_t$) > $b$ then`
10: `v = 1`
11: `else`
12: `v = -1`
13: `trainSamples = [(x[0], x[1], v) for x in trainSamples]`
14: `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 $s_p$

1: totalTrainingQueue = []
2: while stopCounter < $n_{sim}$ do
3:     for $i$ in range($n_{sim}$) do
4:         Initialize MCTS
5:         trainingSamples = getTrainSamples($s_0$)
6:         if len(totalTrainingQueue) > $L$ then
7:             totalTrainingQueue.pop()
8:         totalTrainingQueue += trainingSamples
9:         $f_\theta$ = RLTrain(totalTrainingQueue, $f_\theta$)
10:    Get the slimmed network $s'$ by pruning $s_0$ with $f_\theta$
11:    if trainAcc($s'$) > $b$ then
12:        $b = \text{trainAcc}(s')$
13:        $s_p = s'$
14:        stopCounter = 0
15:    else
16:        stopCounter += 1
17:    return $s_p$