Runtime Neural Pruning

Ji Lin*, Yongming Rao*, Jiwen Lu
Tsinghua University

Abstract

In this paper, we propose a **Runtime Neural Pruning (RNP)** framework which prunes the deep neural network **dynamically at the runtime**.

Unlike existing neural pruning methods which produce a fixed pruned model for deployment, our method preserves the **full ability** of the original network and conducts pruning according to the input image and current feature maps **adaptively**.

The pruning is performed in a **bottom-up, layer-by-layer** manner, which we model as a **Markov decision process** and use reinforcement learning for training.

Since the ability of network is fully preserved, the balance point is easily **adjustable**.

RNP

In this section, we introduce the details of our RNP framework.

1. **Bottom-up Runtime Pruning**

backbone CNN C with conv layers $C_1, C_2, \dots, C_m$, corresponding kernels $\boldsymbol{K}_1, \boldsymbol{K}_2, \dots, \boldsymbol{K}_m$, #channels n_i, producing feature maps $\boldsymbol{F}_1, \boldsymbol{F}_2, \dots, \boldsymbol{F}_m$, with size $n_i \times H \times W$.

Goal: find and prune the redundant convolutional kernels in $\boldsymbol{K}_{i+1}$, given feature maps $\boldsymbol{F}_i, i = 1, 2, \dots, m-1$, to reduce computation and achieve maximum performance simultaneously.

$$\min_{\mathbf{K}_{i+1}, h} \mathbb{E}_{\mathbf{F}_i}[L_{cls}(\mathrm{conv}(\mathbf{F}_i, \mathbf{K}[h(\mathbf{F}_i)])) + L_{pnt}(h(\mathbf{F}_i))],$$

L_{cls} - classification loss, L_{pnt} - computation penalty

2. **Layer-by-layer Markov Decision Process**

State: Given feature map $\boldsymbol{F}_i$, extract dense feature embedding $p_{\boldsymbol{F}_i}$ with global pooling, and use a encoder E, to project into a fixed length embedding $E(p_{\boldsymbol{F}_i})$.

Action: actions for each pruning are defined in an incremental way: taking actions a_i yields calculating the feature map groups $\boldsymbol{F}'_1, \boldsymbol{F}'_2, \dots, \boldsymbol{F}'_i, i = 1,2, \dots, k$.

Reward: The reward of each action taken at the t-th step with action a_i is defined as:

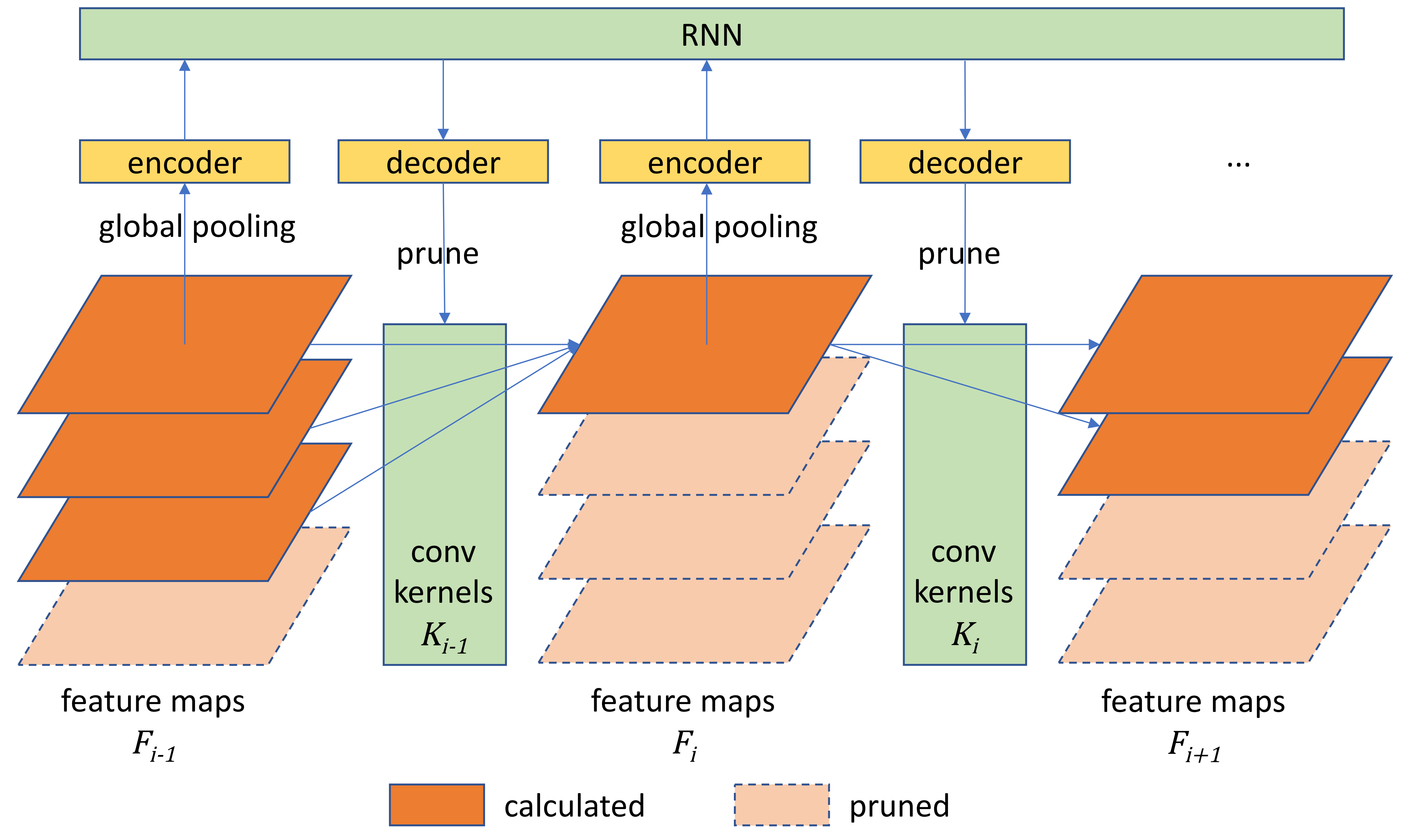
$$r_t(a_i) = \begin{cases} -\alpha L_{cls} + (i-1) \times p, & \text{if inference terminates } (t = m-1), \\ (i-1) \times p, & \text{otherwise } (t < m-1) \end{cases}$$

here p is a negative penalty.

Loss: $\min_{\theta} L_{re} = \mathbb{E}[r(s_t, a_i) + \gamma \max_{a_i} Q(s_{t+1}, a_i) - Q(s_t, a_i)]^2,$

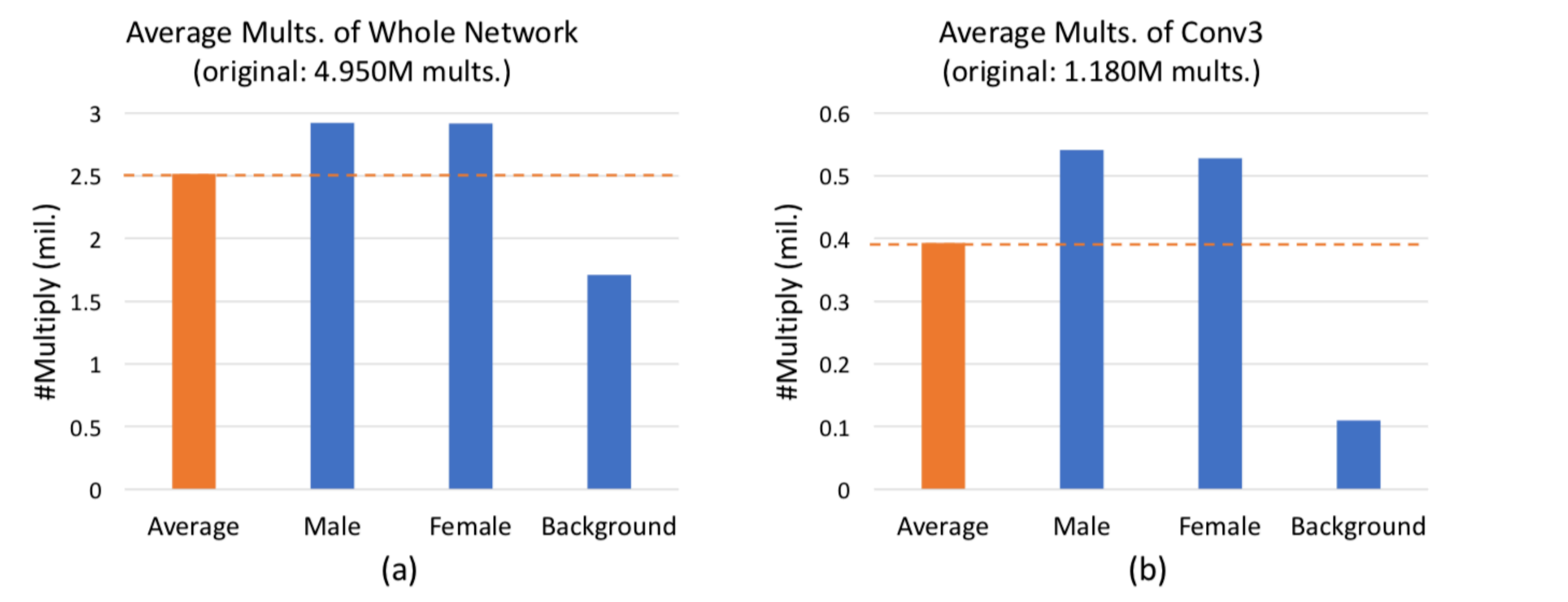
Overall Framework

RNP consists of two sub-networks: the backbone CNN net- work and the decision network. The convolution kernels of backbone CNN network are dynamically pruned according to the output Q-value of decision network.

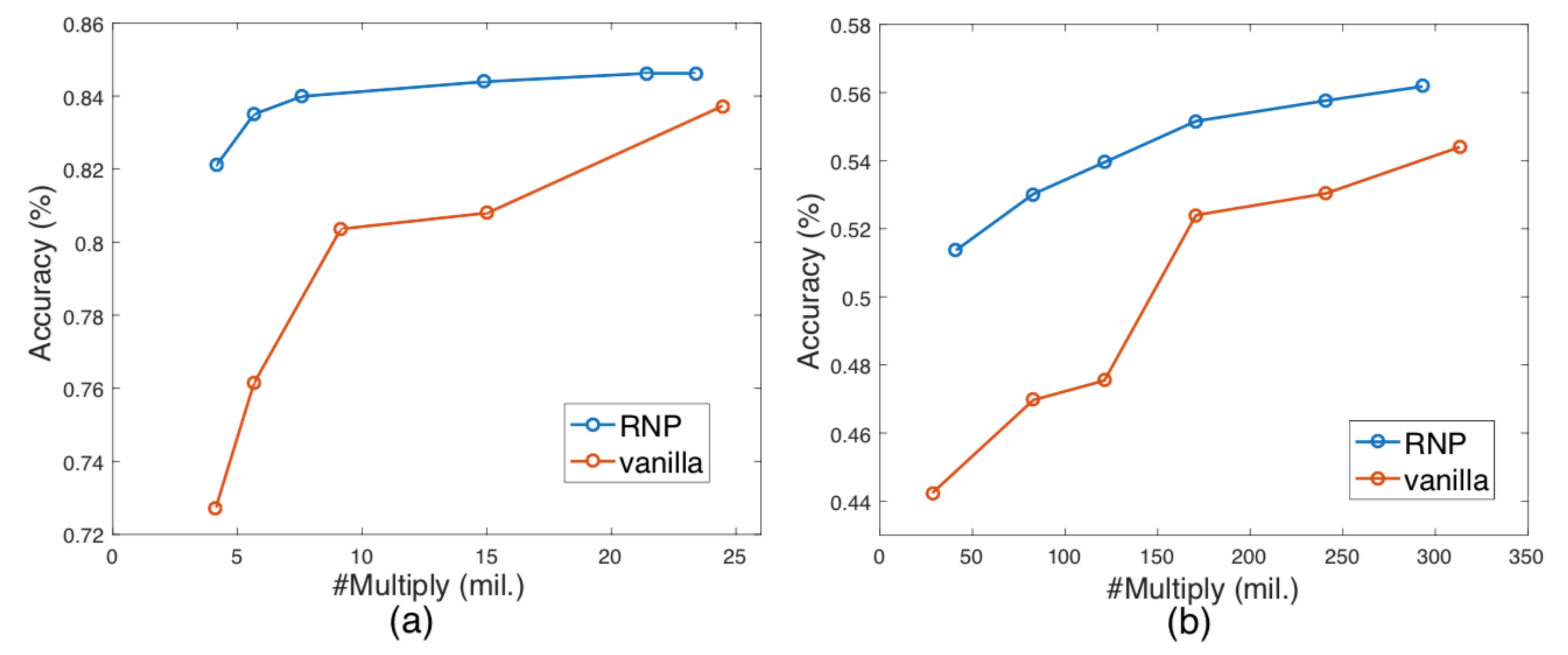


Experiment Results

- Intuitive 3-class classification experiment on *LFW-T*



- Results on CIFAR10 and CIFAR-100



- Comparisons of increase of top-5 error on ILSVRC2012-val (%) with recent state-of-the-arts. *(base top-5 error: 10.1%)*

Speed-up	3×	4×	5×	10×
Jaderberg *et al.* [16] ([40]'s implementation)	2.3	9.7	29.7	
Asymmetric [40]	-	3.84	-	
Filter pruning [22] (our implementation)	3.2	8.6	14.6	
Ours	**2.32**	**3.23**	**3.58**	**4.89**

- GPU inference time under different theoretical speed-up ratios on ILSVRC2012-val set.

Speed-up solution	Increase of top-5 error (%)	Mean inference time (ms)
VGG-16 (1×)	0	3.26 (1.0×)
Ours (3×)	2.32	1.38 (2.3×)
Ours (4×)	3.23	1.07 (3.0×)
Ours (5×)	3.58	0.880 (3.7×)
Ours (10×)	4.89	0.554 (5.9×)

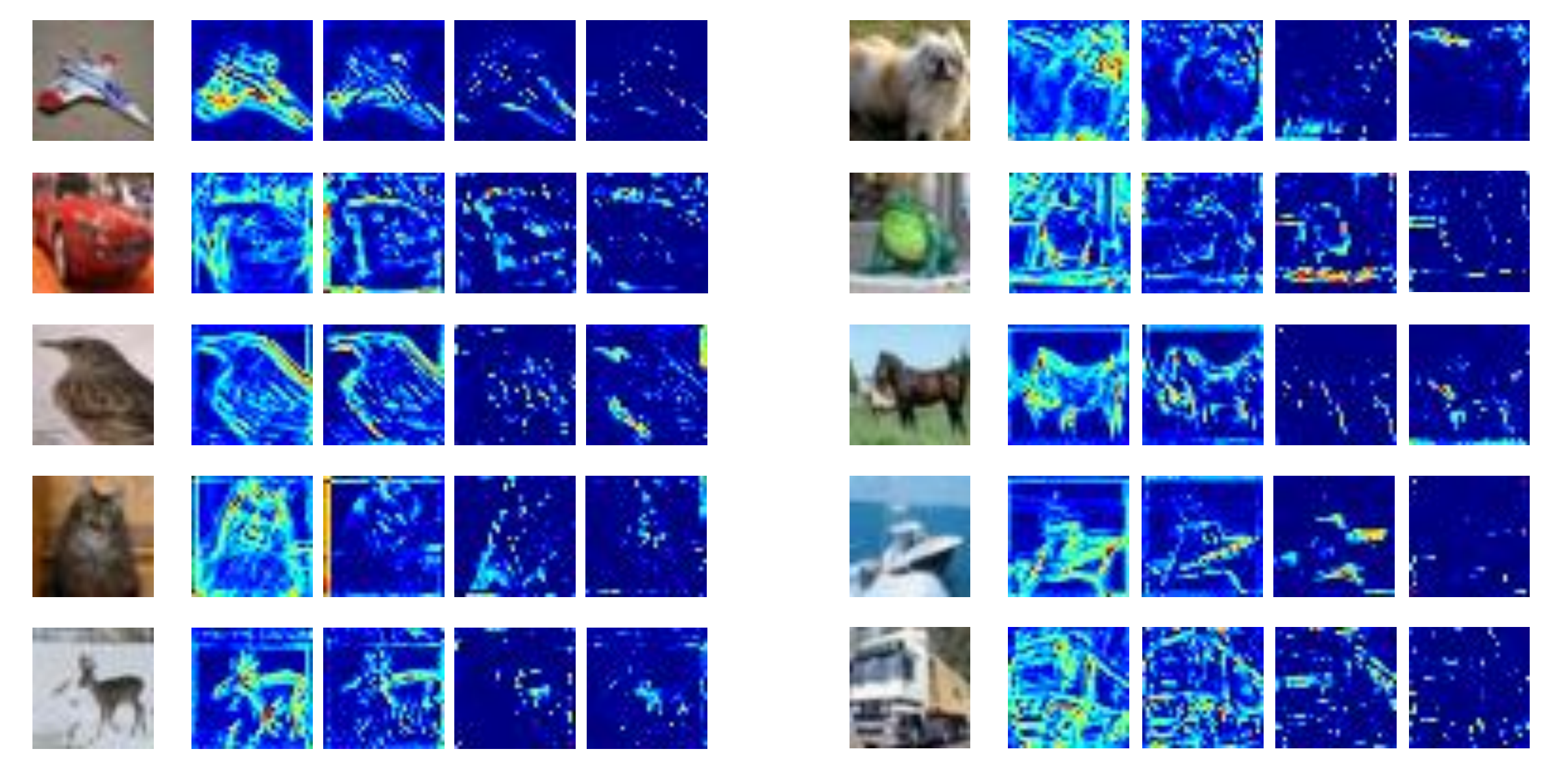
Algorithm

Overall algorithm of RNP.

```
Algorithm 1 Runtime neural pruning for solving optimization problem (1):
Input: training set with labels $\{X\}$
Output: backbone CNN $C$, decision network $D$
 1: initialize: train $C$ in normal way or initialize $C$ with pre-trained model
 2: for $i \leftarrow 1, 2, ..., M$ do
 3:     // train decision network
 4:     for $j \leftarrow 1, 2, ..., N_1$ do
 5:         Sample random minibatch from $\{X\}$
 6:         Forward and sample $\epsilon$-greedy actions $\{s_t, a_t\}$
 7:         Compute corresponding rewards $\{r_t\}$
 8:         Backward $Q$ values for each stage and generate $\nabla_\theta L_{re}$
 9:         Update $\theta$ using $\nabla_\theta L_{re}$
10:     end for
11:     // fine-tune backbone CNN
12:     for $k \leftarrow 1, 2, ..., N_2$ do
13:         Sample random minibatch from $\{X\}$
14:         Forward and calculate $L_{cls}$ after runtime pruning by $D$
15:         Backward and generate $\nabla_C L_{cls}$
16:         Update $C$ using $\nabla_C L_{cls}$
17:     end for
18: end for
19: return $C$ and $D$
```

Feature Map Analysis

Since we define the actions in an incremental way, the convolutional channels of lower index are calculated more, so the functions of different convolution groups might be similar to "low-frequency" and "high- frequency" filters. We visualized different functions of convolutional groups by calculating average feature maps produced by each convolutional group.



Contacts

Ji Lin: *lin-j14@mails.tsinghua.edu.cn*
Yongming Rao: *xxx@xxx*