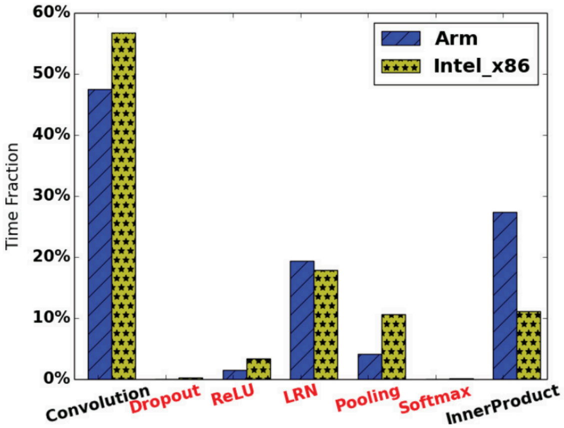
Deeprebirth: Accelerating Deep Neural Network Execution On Mobile Devices

Dawei Li, Xiaolong Wang, Deguang Kong, "Deeprebirth: Accelerating Deep Neural Network Execution On Mobile Devices", in Proc. of the thirty-second AAAI Conference On Artificial Intelligence (AAAI-18)

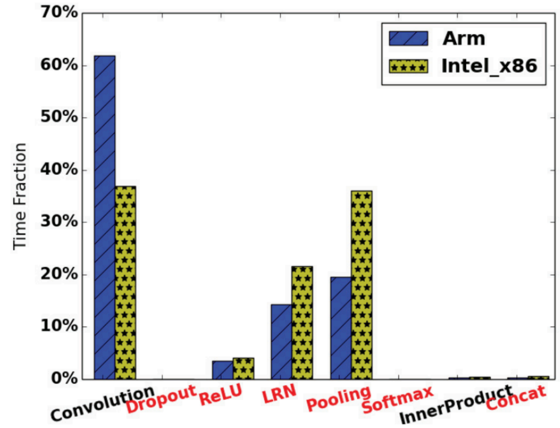
Introduction

- More and more mobile applications adopt deep learning techniques to provide accurate, intelligent and effective services
- Limited resources → execution speed of deep learning models on mobile devices is a bottleneck
- Goal: improving the execution efficiency of deep learning models with minimum accuracy loss

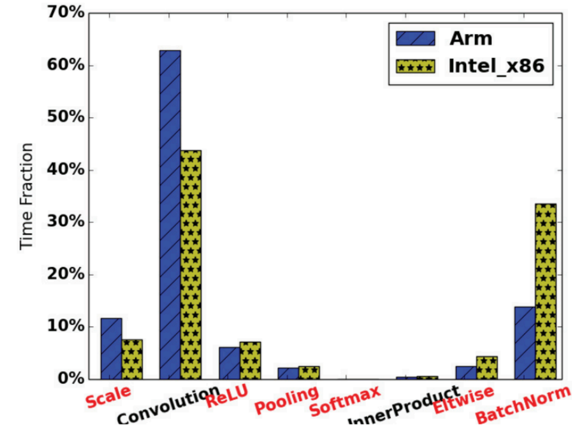
Findings of execution time



(a) AlexNet



(b) GoogLeNet



(c) ResNet-50

- Tensor layer: contain tensor-type parameters, i.e. fully connected layers, convolutional layers

  Non-tensor layer: no contain tensor-type parameters, i.e. ReLU, pooling

Findings of execution time

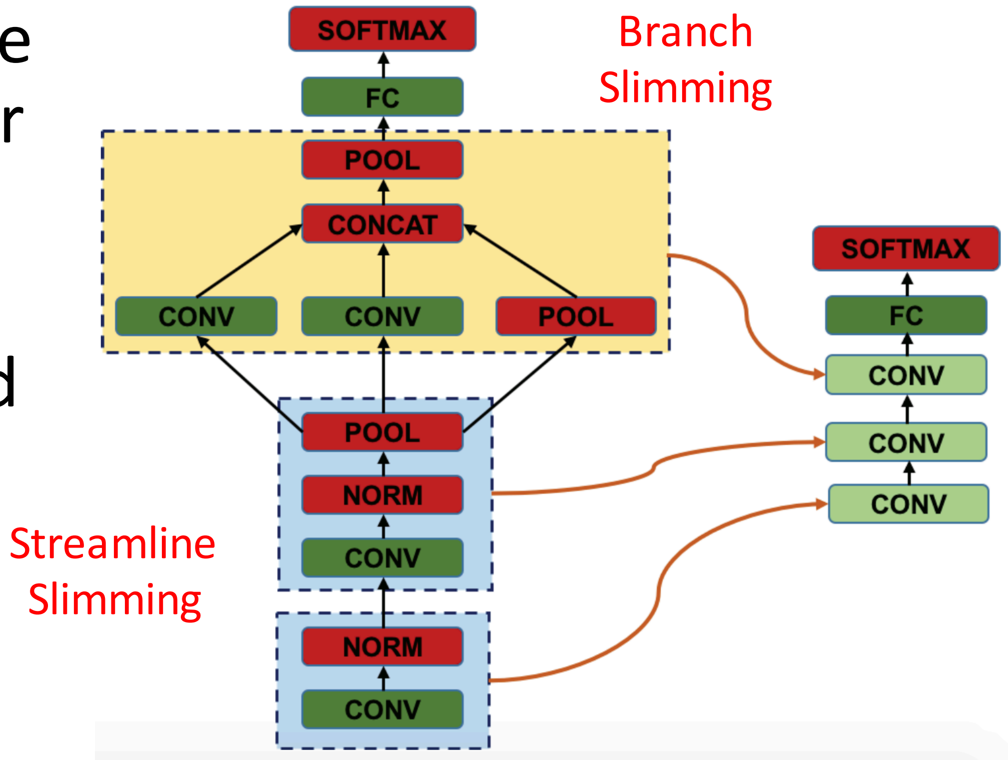
Network	Intel x86	Arm	Titan X
AlexNet	32.08%	25.08%	22.37%
GoogLeNet	62.03%	37.81%	26.14%
ResNet-50	55.66%	36.61%	47.87%
ResNet-152	49.77%	N/A	44.49%
Average	49.89%	33.17%	35.22%

$$\%\ \text{Latency} = \frac{\text{Time spent on Non-tensor layer}}{\text{Time spent over the entire network}},$$

- The execution time of non-tensor layer in Intel x86 CPU is the highest
- Although non-tensor layers do not have as high affect on the mainstream ARM CPUs, on average they still cost about 1/3 of the computing time

Approaches in this paper

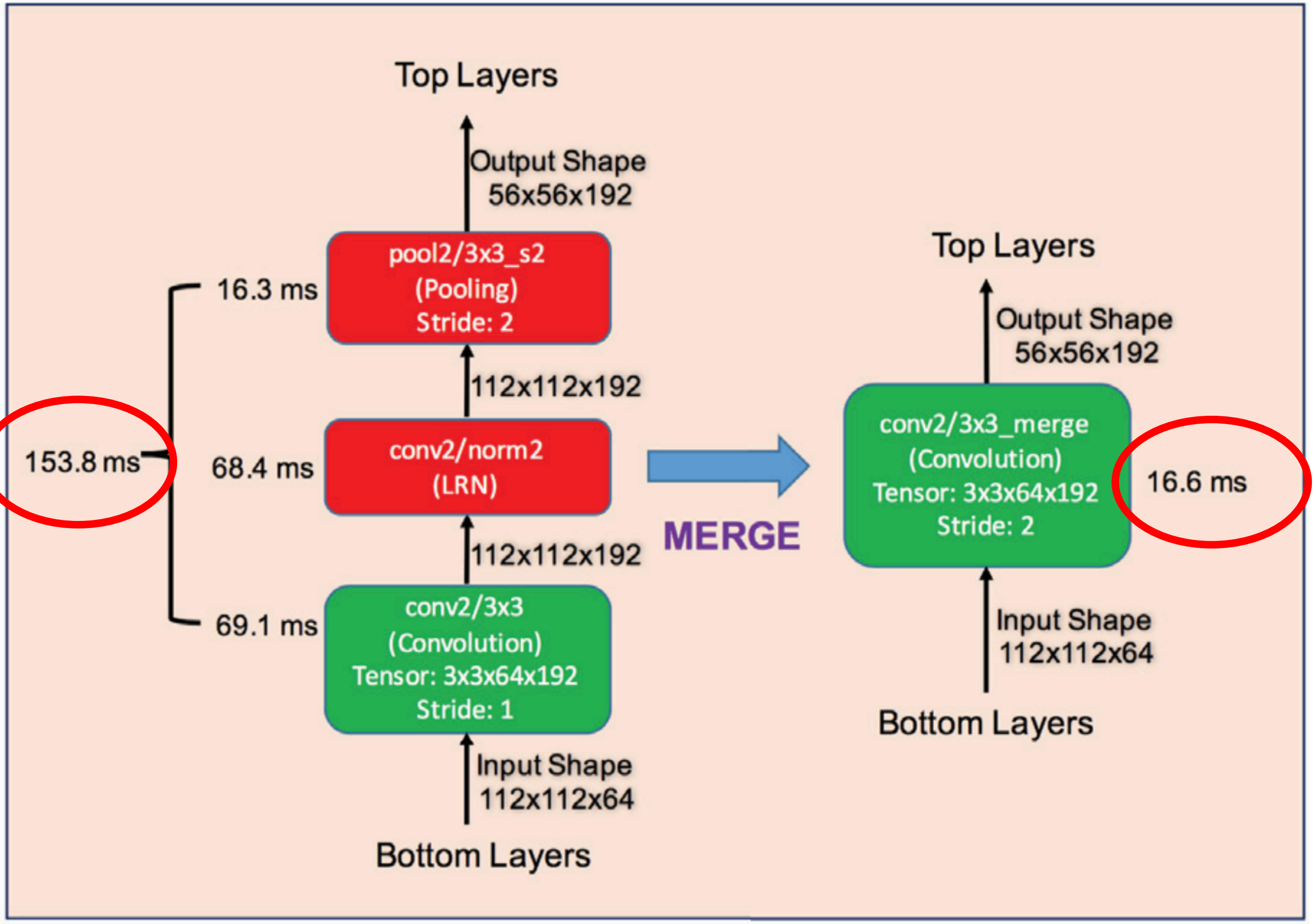
- **Streamline Slimming**: merge the consecutive non-tensor and tensor layer vertically
- **Branch Slimming**: merge non-tensor and tensor branches horizontally



Streamline Slimming

- Observation:
  1. non-tensor layers usually follow a tensor layer such as convolution layer
  2. several consecutive layers can be viewed as a black box and can be replaced by a new tensor-layer by parameter learning
- Method:
  - **Pooling Layer**: remove the pooling layer and set the stride value of the new convolution layer as the product of the stride values for both the original pooling layer and the convolution layer
  - **Non-Pooling Layer**: directly prune those layers from the original deep neural network

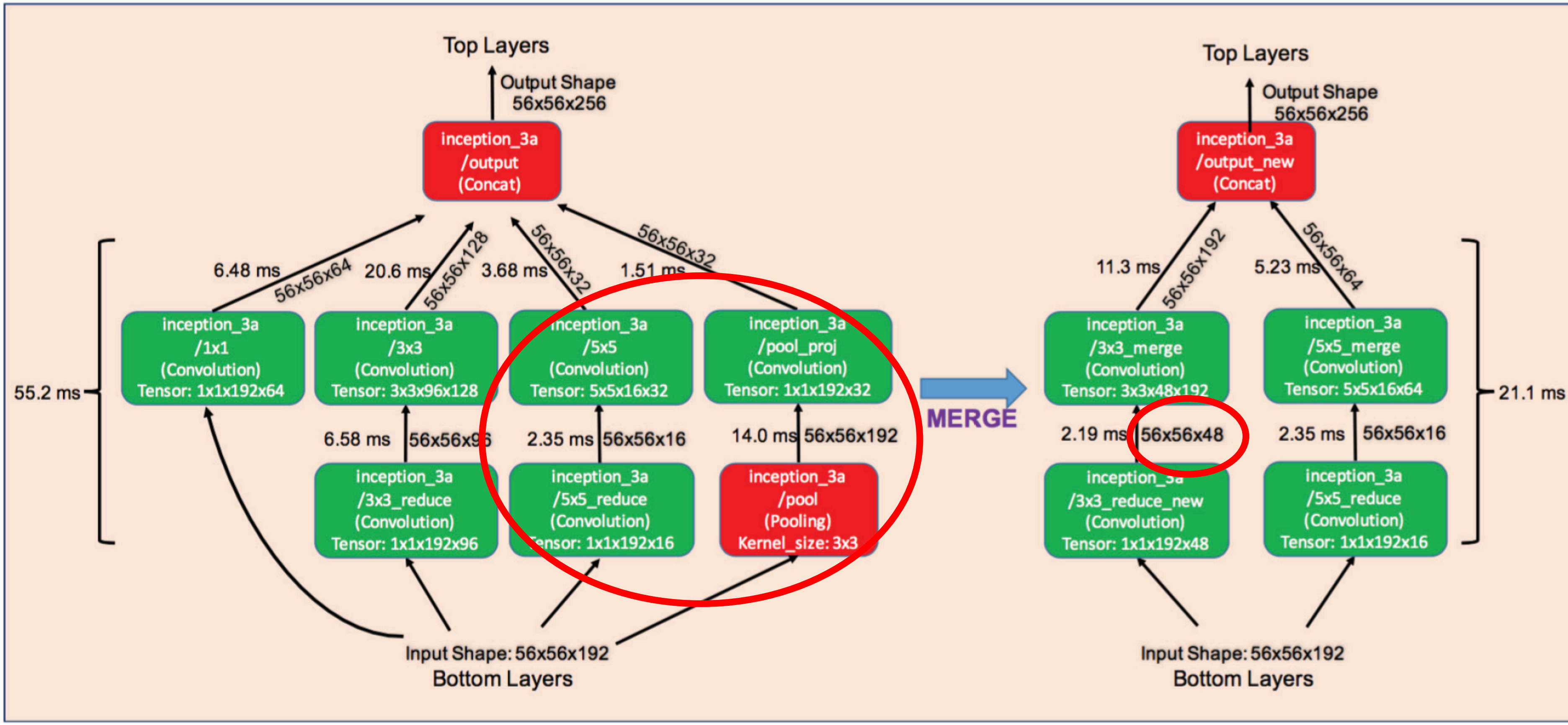
Streamline Slimming: example



Branch Slimming

- Observation:
  - GoogLeNet has 4 branches: 3 convolution branches take feature maps from the bottom layer at various scales (1x1 , 3x3 and 5x5) and one 3x3 pooling branch
- Method:
  - recreate a new tensor layer (i.e., slim layer) by fusing the non-tensor branch and a tensor unit with relatively small latency to output the feature maps that were originally generated by the non-tensor branch
  - the picked tensor branch's kernel size should be at least the size of the non-tensor branch
  - reduce feature maps channels

Branch Slimming: example



Retraining

- Set the learning rate of new layers 10 times over those in other layers

Evaluation: GoogLeNet

- Accuracy:

Step	Slim Layer(s)	Top-5 Accuracy
0	N/A	88.89%
1	conv1	88.73%
2	conv2	88.82%
3	inception_3a	88.50%
4	inception_3b	88.27%
5	inception_4a	88.60%
6	inception_4b-4d	88.61%
7	inception_4e	88.43%
8	inception_5a	88.41%
9	inception_5b	**88.43%**
Tucker Decomposition	**ALL**	**86.54%**

Evaluation: GoogLeNet

- Speed (different layer):

Step	Slim Layer(s)	Top-5 Accuracy
0	N/A	88.89%
1	conv1	88.73%
2	conv2	88.82%
3	inception_3a	88.50%
4	inception_3b	88.27%
5	inception_4a	88.60%
6	inception_4b-4d	88.61%
7	inception_4e	88.43%
8	inception_5a	88.41%
9	inception_5b	**88.43%**
Tucker Decomposition	**ALL**	**86.54%**

Evaluation: GoogLeNet

- Speed (different method):

Device	GoogLeNet	GoogLeNet -Tucker	GoogLeNet -Slim	GoogLeNet -Slim-Tucker	SqueezeNet
Moto E	1168.8 ms	897.9 ms	406.7 ms	**213.3 ms**	291.4 ms
Samsung Galaxy S5	651.4 ms	614.9 ms	210.6 ms	**106.3 ms**	136.3 ms
Samsung Galaxy S6	424.7 ms	342.5 ms	107.7 ms	**65.34 ms**	75.34 ms
Macbook Pro (CPU)	91.77 ms	78.22 ms	23.69 ms	**15.18 ms**	17.63 ms
Titan X	10.17 ms	10.74 ms	6.57 ms	7.68 ms	**3.29 ms**

Evaluation: GoogLeNet

- Storage and memory:

Model	**Energy**	**Storage**	**Memory**	**Max Batch Size on Titan X**
GoogLeNet	984 mJ	26.72 MB	33.2 MB	350
GoogLeNet-Tucker	902 mJ	14.38 MB	35.8 MB	323
GoogLeNet-Slim	**447 mJ (2.2x)**	23.77 MB	13.2 MB	**882 (2.52x)**
GoogLeNet-Slim-Tucker	**226 mJ (4.4x)**	11.99 MB	14.8 MB	**785 (2.24x)**
SqueezeNet	288 mJ	4.72 MB	36.5 MB	321

Evaluation: AlexNet

Step	Slim Layer(s)	Top-5 Accuracy	Speed-up	Energy Cost
0	N/A	80.03%	445 ms	688 mJ
1	conv1+norm1 $\rightarrow$ conv1	79.99%	343 ms (1.29x)	555 mJ (1.24x)
2	conv2+norm2 $\rightarrow$ conv2	79.57%	274 ms (1.63x)	458 mJ (1.51x)

Evaluation: ResNet

Step	Slim Layer(s)	Top-5 Accuracy	Speed-up	Runtime-Mem Batch32
0	N/A	92.36%	189 ms	2505 MB
1	conv1	92.13%	162 ms (1.17x)	2113 MB (1.19x)
2	res2a_branch1	92.01%	140 ms (1.35x)	1721 MB (1.46x)
3	res2a_branch2a-2c	91.88%	104 ms (1.82x)	1133 MB (2.21x)

Conclusion

- DeepRebirth is proposed to speed up the neural networks with satisfactory accuracy, which operates by re-generating new tensor layers from optimizing non-tensor layers and their neighborhood units
- Future work: integrate DeepRebirth with other state-of-the-art tensor layer compression methods and also extend our evaluation to heterogeneous mobile processors