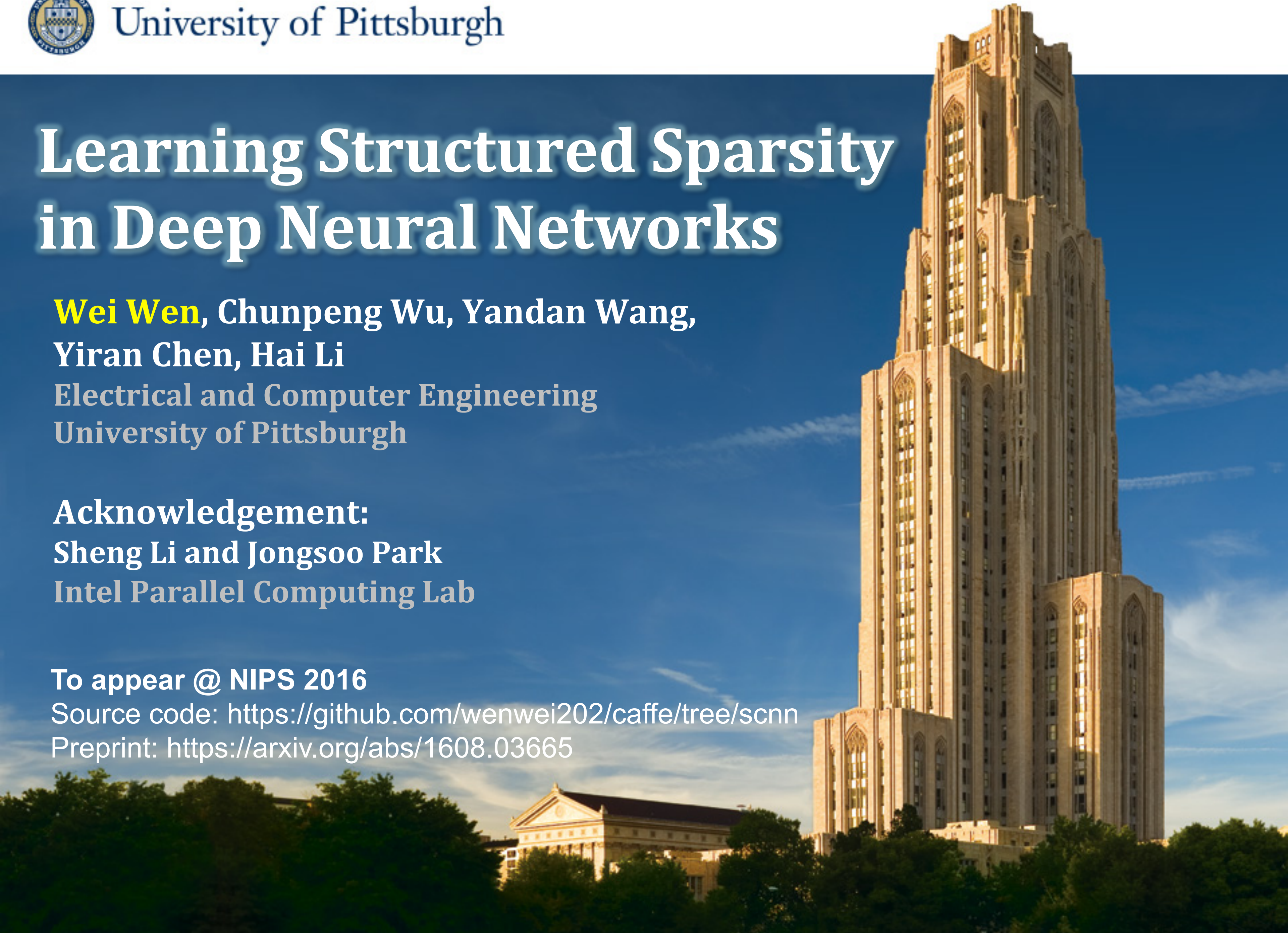
University of Pittsburgh



Learning Structured Sparsity in Deep Neural Networks

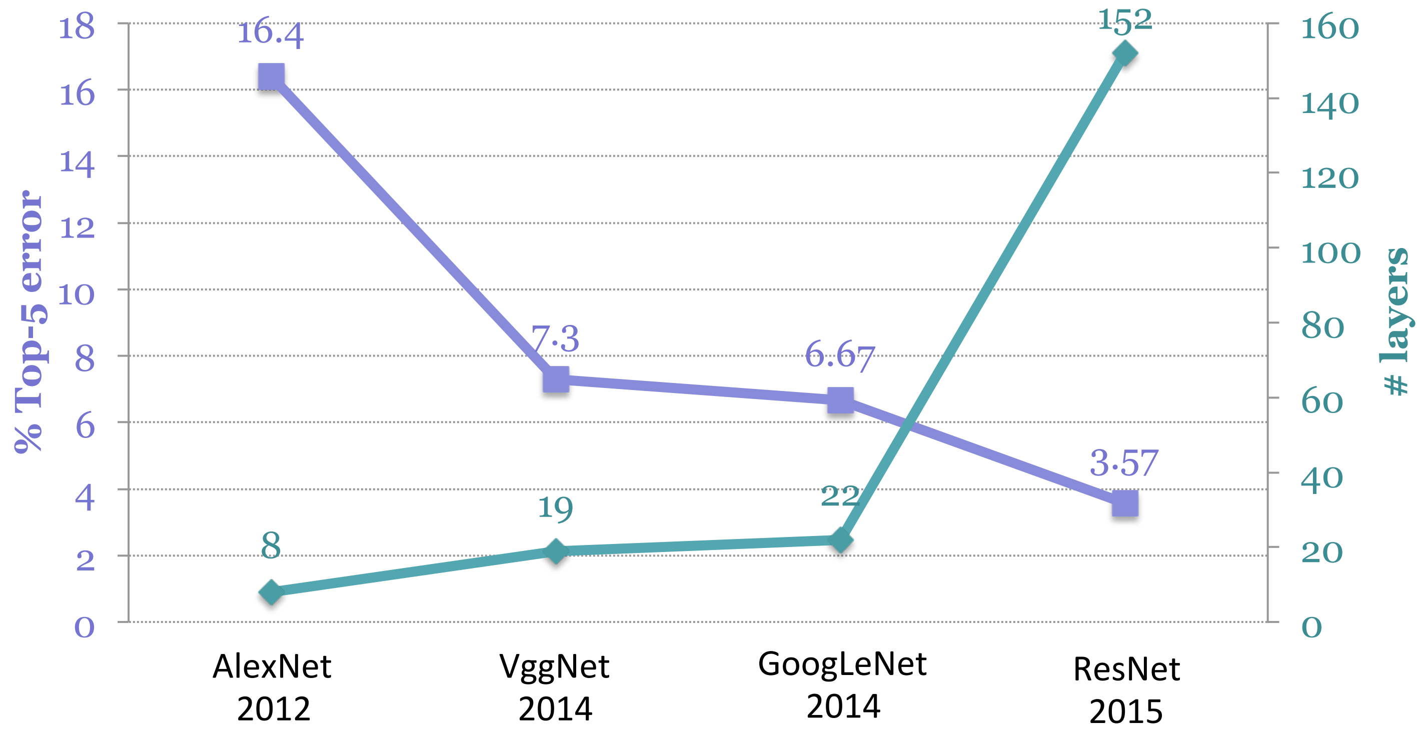
Wei Wen, Chunpeng Wu, Yandan Wang,
Yiran Chen, Hai Li
Electrical and Computer Engineering
University of Pittsburgh

Acknowledgement:
Sheng Li and Jongsoo Park
Intel Parallel Computing Lab

To appear @ NIPS 2016
Source code: https://github.com/wenwei202/caffe/tree/scnn
Preprint: https://arxiv.org/abs/1608.03665

Deeper Neural Networks

Deeper neural networks (DNN) with lower classification error

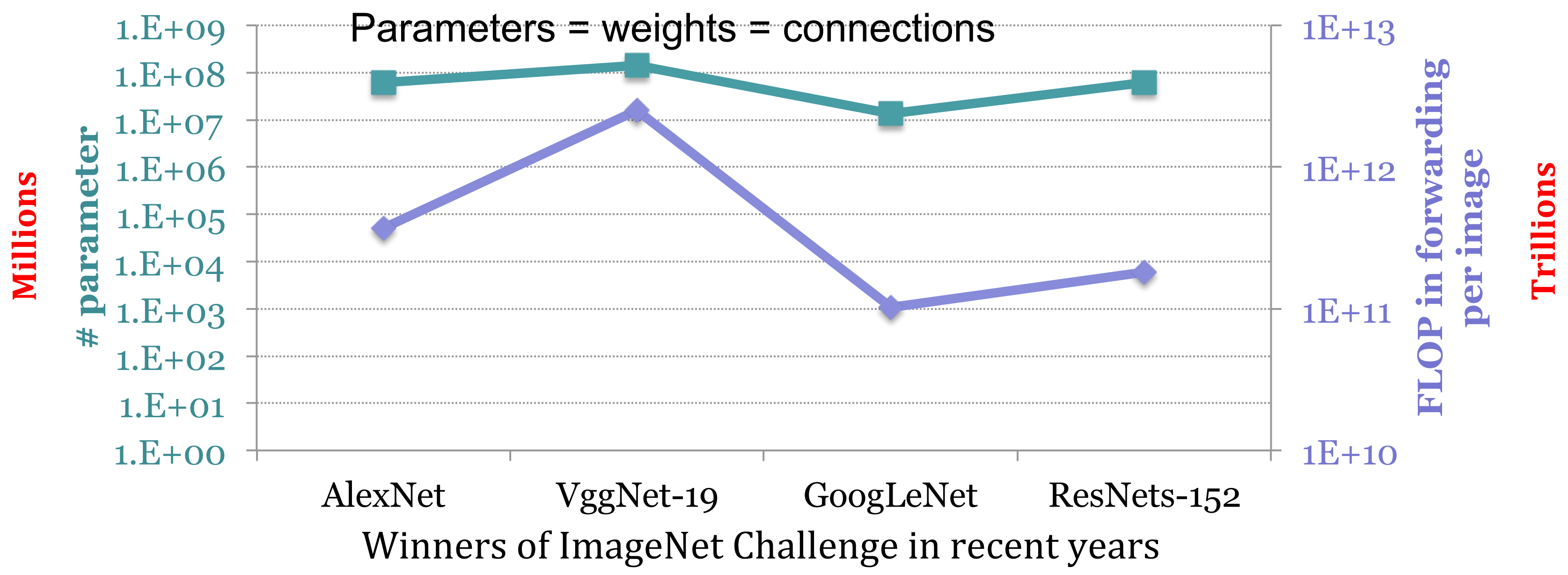


Winners of ImageNet Challenge in recent years

Deeper neural networks are the trend but burden computation in modern hardware!

Complexity of Deep Neural Networks

Fewer parameters, fewer computation (FLOP: Floating Point Operation)



How to reduce the number of parameters in DNN so as to reduce FLOP, meanwhile maintain the classification accuracy?

Related Works

- State-of-the-art methods to reduce the number of parameters
  - Weight regularization (L1-norm)
  - Connection pruning

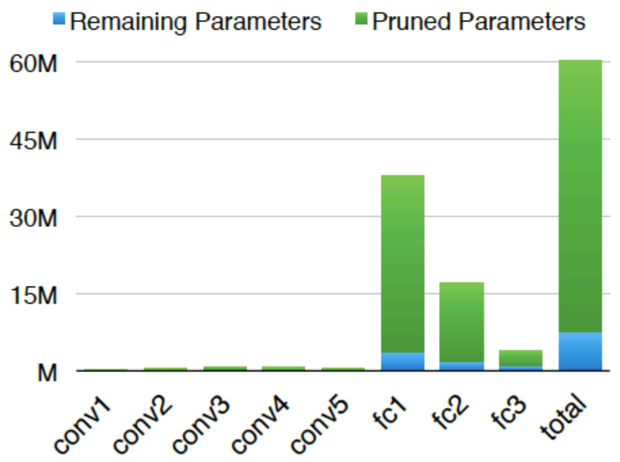
Sparsity: the ratio of zeros

AlexNet, B. Liu, *et al.*, CVPR 2015

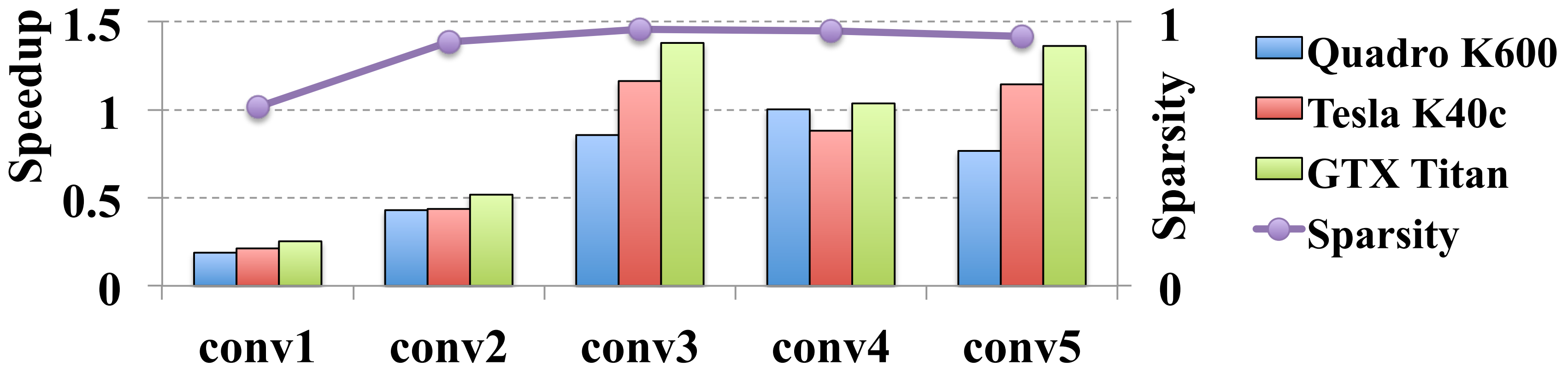
Layer	conv1	conv2	conv3	conv4	conv5
Sparsity%	0.927	0.95	0.951	0.942	0.938
Theoretical speedup	2.61	7.14	16.12	12.42	10.77

AlexNet, S. Han, *et al.*, NIPS 2015

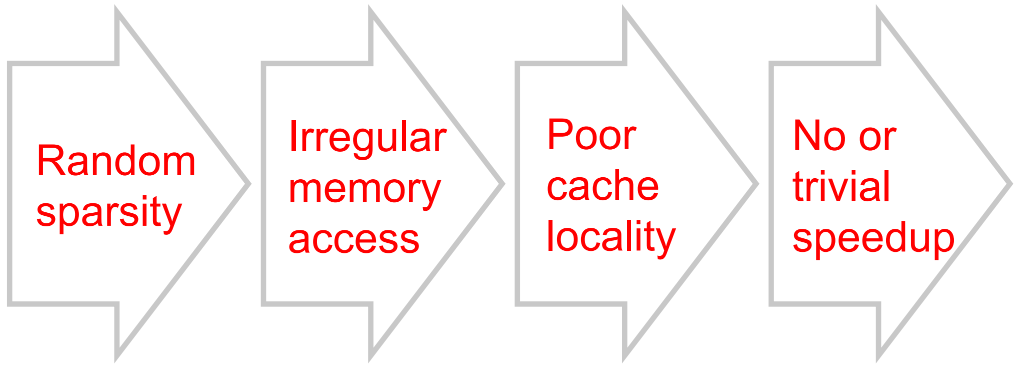
Layer	Weights	FLOP	Act%	Remained Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
Total	61M	1.5B	54%	**11%**	**30%**



Theoretical Speedup ≠ Practical Speedup



Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.



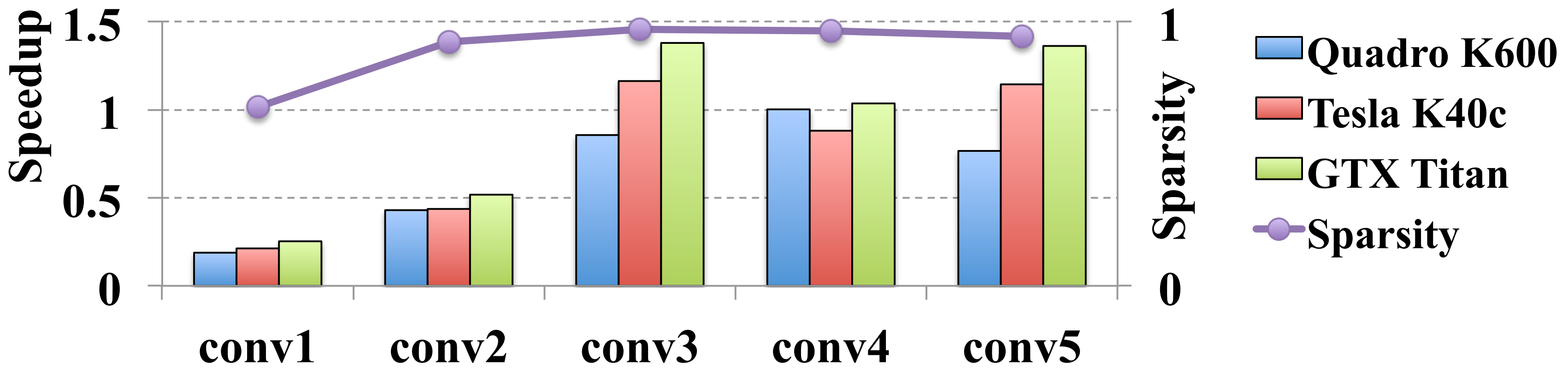
Hardcoding nonzero weights in source code in B. Liu, etc., CVPR 2015

Software customization

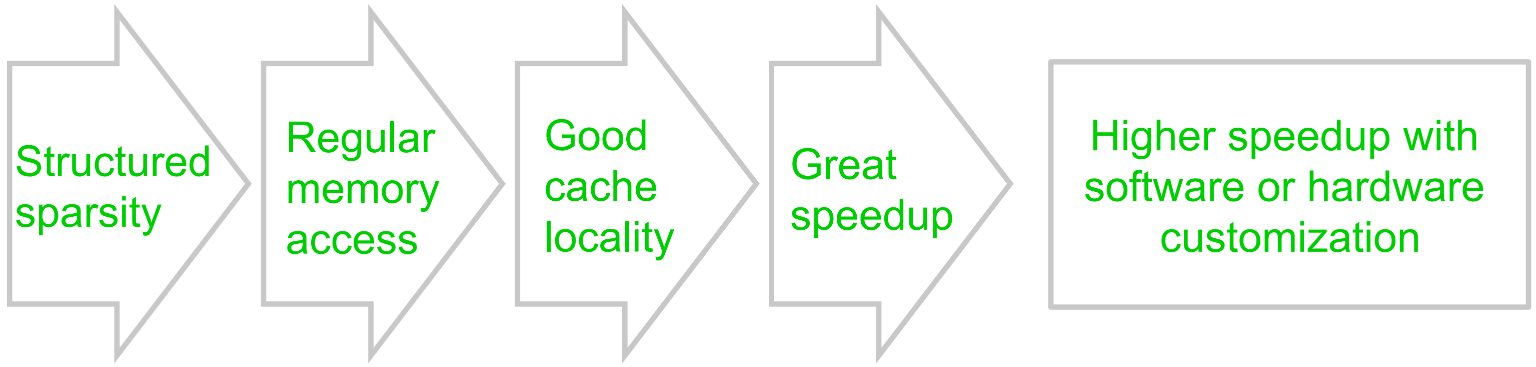
Hardware customization

Customizing an EIE chip accelerator for compressed DNN in S. Han ISCA 2017

Theoretical Speedup ≠ Practical Speedup

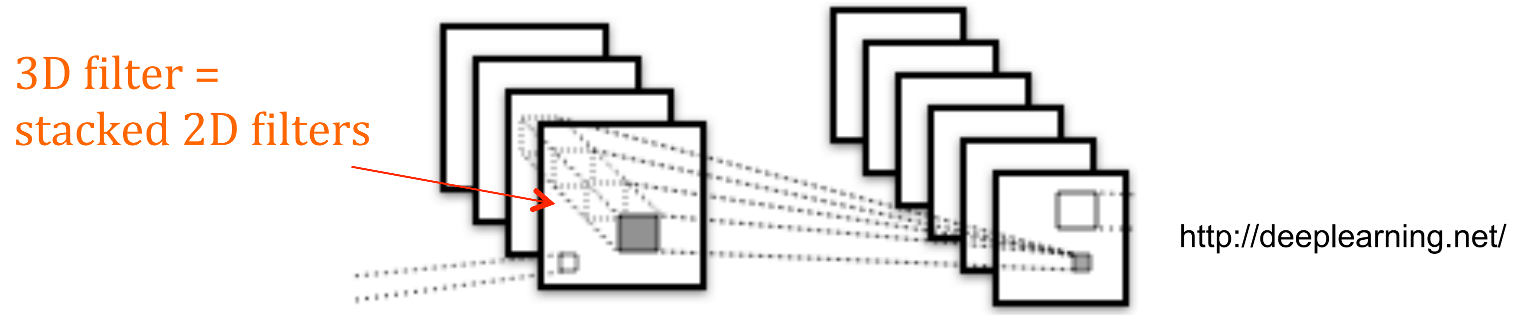


Forwarding speedups of AlexNet on GPU platforms and the sparsity. Baseline is GEMM of cuBLAS. The sparse matrixes are stored in the format of Compressed Sparse Row (CSR) and accelerated by cuSPARSE.

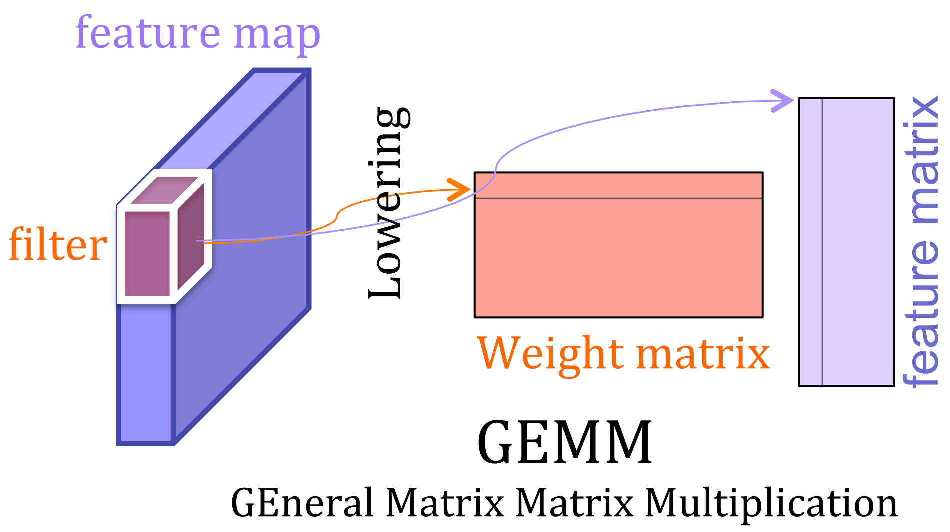


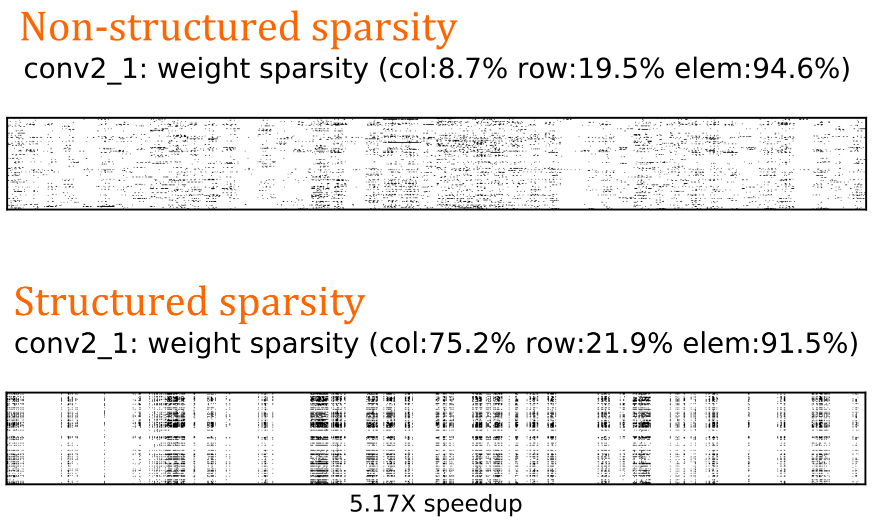
Computation-efficient Structured Sparsity

Example 1: Removing 2D filters in convolution (2D-filter-wise sparsity)



Example 2: Removing rows/columns in GEMM (row/column-wise sparsity)





Structured Sparsity Regularization

- Group Lasso regularization in ML model

$$\arg\min_{\mathbf{w}}\left\{E\left(\mathbf{w}\right)\right\}=\arg\min_{\mathbf{w}}\left\{E_D\left(\mathbf{w}\right)+\lambda_g\cdot R_g\left(\mathbf{w}\right)\right\}$$

Many groups will be zeros

$$R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g,$$

$$||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|}\left(w_i^{(g)}\right)^2}$$

Structured Sparsity Regularization

- Group Lasso regularization in ML model

$$\arg\min_{\mathbf{w}}\left\{E(\mathbf{w})\right\}=\arg\min_{\mathbf{w}}\left\{E_D(\mathbf{w})+\lambda_g\cdot R_g(\mathbf{w})\right\}$$

$$\Updownarrow$$

$$\arg\min_{\mathbf{w}}\left\{E(\mathbf{w})\right\}=\arg\min_{\mathbf{w}}\left\{E_D(\mathbf{w})\right\}$$

$$s.t.\ R_g(\mathbf{w})\le\eta_g$$

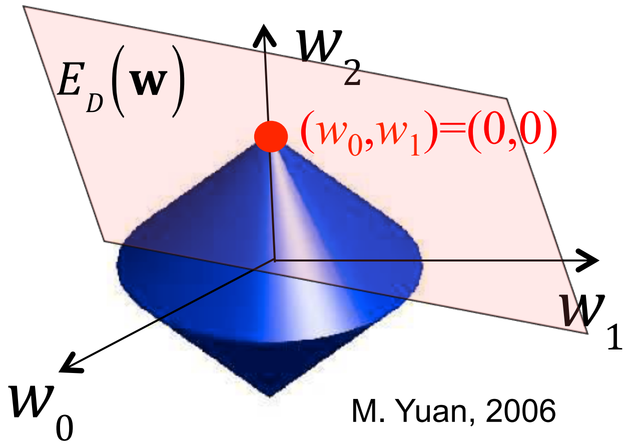
Many groups will be zeros

$$R_g(\boldsymbol{w}) = \textstyle\sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g,$$

$$||\boldsymbol{w}^{(g)}||_g = \sqrt{\textstyle\sum_{i=1}^{|\boldsymbol{w}^{(g)}|}\left(w_i^{(g)}\right)^2}$$

Example:

$$R_g\Big(\underbrace{\boxed{w_0,w_1}}_{\text{group 1}},\underbrace{\boxed{w_2}}_{\text{group 2}}\Big)=\sqrt{w_0^2+w_1^2}+\sqrt{w_2^2}\le\eta_g$$



M. Yuan, 2006

SSL: Structured Sparsity Learning

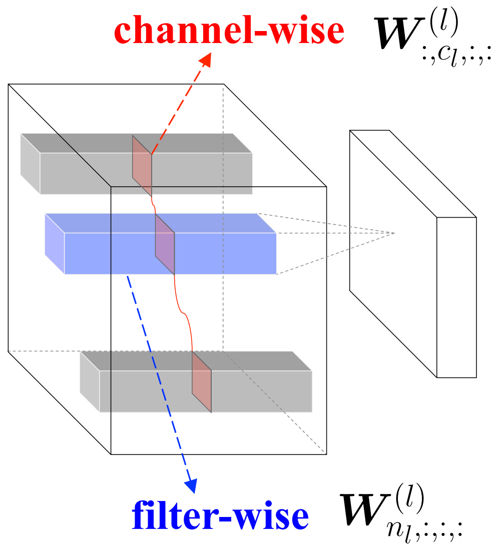
- Group Lasso regularization in DNNs

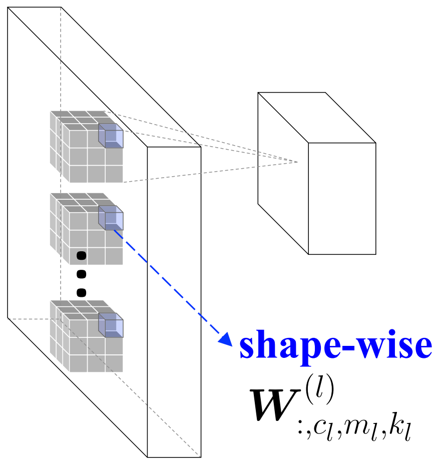
$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda \cdot R(\boldsymbol{W}) + \lambda_g \cdot \sum_{l=1}^{L} R_g\left(\boldsymbol{W}^{(l)}\right)$$

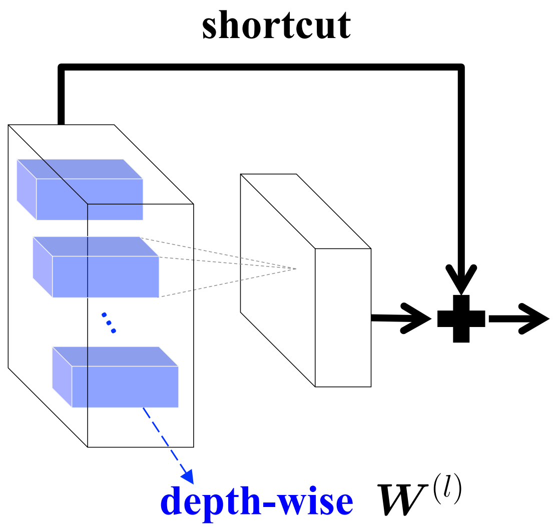
$$R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g$$

Learned structured sparsity is determined by the way of splitting groups

Penalize unimportant filters and channels | **Learn filter shapes** | **Learn the depth of layers**

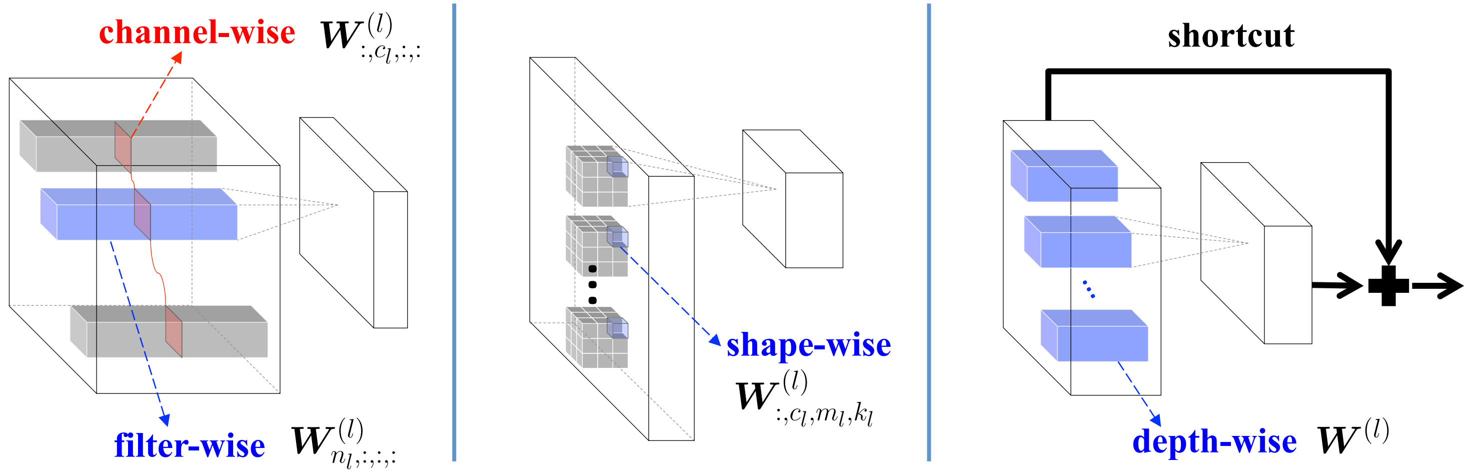

channel-wise $\boldsymbol{W}^{(l)}_{:,c_l,:,:}$
filter-wise $\boldsymbol{W}^{(l)}_{n_l,:,:,:}$





SSL: Structured Sparsity Learning

- Group Lasso regularization in DNNs



channel-wise $W^{(l)}_{:,c_l,:,:}$
filter-wise $W^{(l)}_{n_l,:,:,:}$
shape-wise $W^{(l)}_{:,c_l,m_l,k_l}$
shortcut
depth-wise $W^{(l)}$

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda_n \cdot \sum_{l=1}^{L} \left(\sum_{n_l=1}^{N_l} ||\boldsymbol{W}^{(l)}_{n_l,:,:,:}||_g \right) + \lambda_c \cdot \sum_{l=1}^{L} \left(\sum_{c_l=1}^{C_l} ||\boldsymbol{W}^{(l)}_{:,c_l,:,:}||_g \right).$$

$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda_s \cdot \sum_{l=1}^{L} \left(\sum_{c_l=1}^{C_l} \sum_{m_l=1}^{M_l} \sum_{k_l=1}^{K_l} ||\boldsymbol{W}^{(l)}_{:,c_l,m_l,k_l}||_g \right).$$

$$E(\boldsymbol{W}) \ = \ E_D(\boldsymbol{W}) + \lambda_d \cdot \textstyle\sum_{l=1}^{L} ||\boldsymbol{W}^{(l)}||_g.$$

Experiments – Penalizing unimportant filters and channels

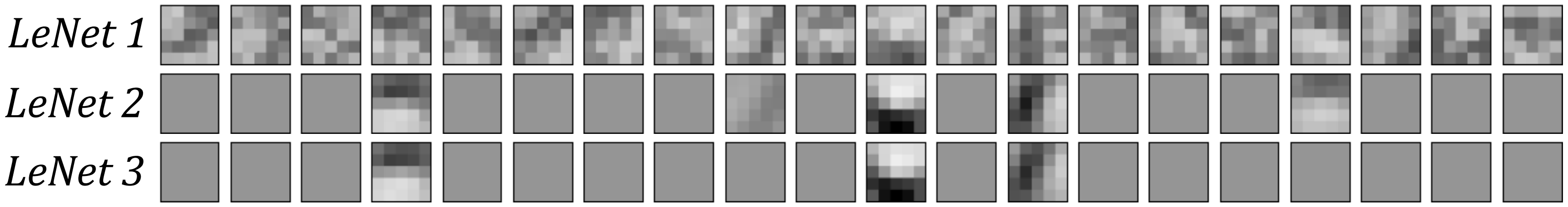
LeNet on MNIST

Table 1: Results after penalizing unimportant filters and channels in *LeNet*

LeNet #	Error	Filter # [§]	Channel # [§]	FLOP [§]	Speedup [§]
1 (*baseline*)	0.9%	20—50	1—20	100%—100%	1.00×—1.00×
2	0.8%	5—19	1—4	25%—7.6%	1.64×—5.23×
3	1.0%	3—12	1—3	15%—3.6%	1.99×—7.44×

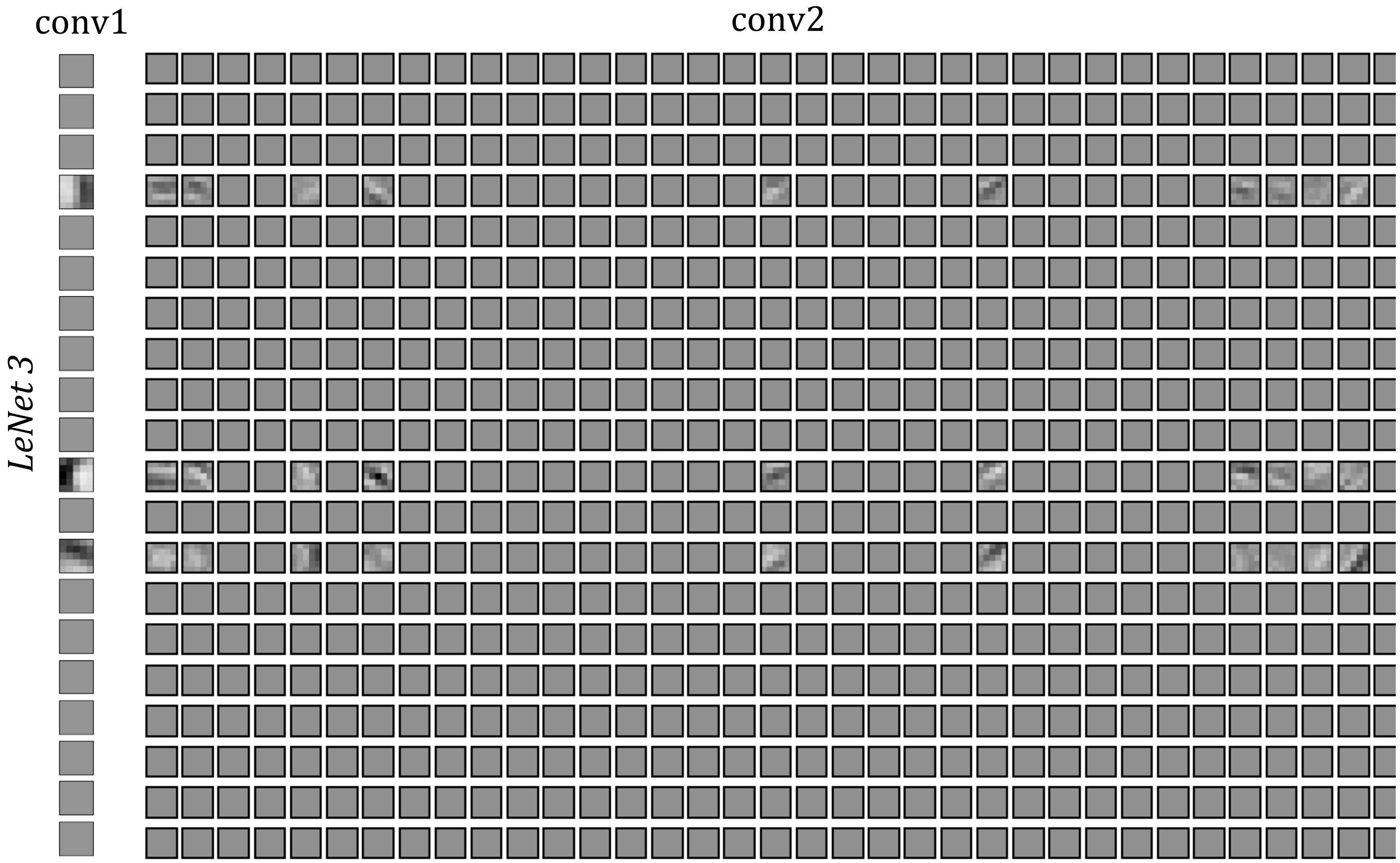
[§] In the order of *conv1*—*conv2*

conv1 filters (gray level 128 represents zero)



Fewer but more natural patterns

Experiments – Penalizing unimportant filters and channels



SSL can efficiently learn DNNs with fewer filters and channels without accuracy loss

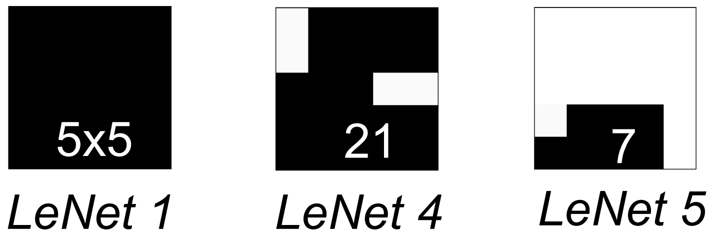
Experiments – Learning smaller filter shapes

Table 2: Results after learning filter shapes in *LeNet*

LeNet #	Error	Filter size [§]	Channel #	FLOP	Speedup
1 (*baseline*)	0.9%	25—500	1—20	100%—100%	1.00×—1.00×
4	0.8%	21—41	1—2	8.4%—8.2%	2.33×—6.93×
5	1.0%	7—14	1—1	1.4%—2.8%	5.19×—10.82×

[§] The sizes of filters after removing zero shape fibers, in the order of *conv1—conv2*

Learned shapes of conv1 filters:



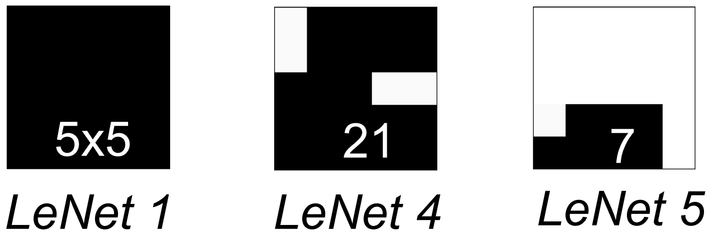
Experiments – Learning smaller filter shapes

Table 2: Results after learning filter shapes in *LeNet*

LeNet #	Error	Filter size §	Channel #	FLOP	Speedup
1 (*baseline*)	0.9%	25—500	1—20	100%—100%	1.00×—1.00×
4	0.8%	21—41	1—2	8.4%—8.2%	2.33×—6.93×
5	1.0%	7—14	1—1	1.4%—2.8%	5.19×—10.82×

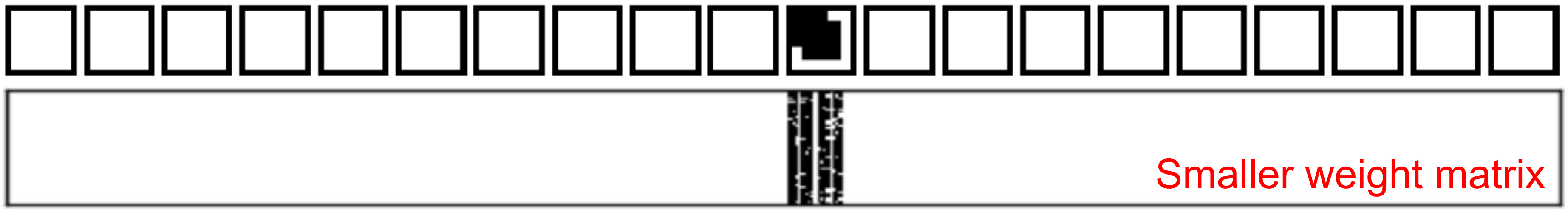
§ The sizes of filters after removing zero shape fibers, in the order of *conv1*—*conv2*

Learned shapes of conv1 filters:



Learned shape of conv2 filters @ *LeNet 5*

3D 20x5x5 filters is regularized to 2D filters!



SSL can efficiently learn DNNs with smaller filters without accuracy loss

Experiments – Learning smaller dense weight matrix

Filter-wise sparsity = row-wise sparsity
Shape-wise sparsity = column-wise sparsity $\longrightarrow$ Smaller dense weight matrix

Table 3: Learning row-wise and column-wise sparsity of *ConvNet* on CIFAR-10

ConvNet #	Error	Row sparsity §	Column sparsity §	Speedup §
1 (*baseline*)	17.9%	12.5%–0%–0%	0%–0%–0%	1.00×–1.00×–1.00×
2	17.9%	50.0%–28.1%–1.6%	0%–59.3%–35.1%	1.43×–3.05×–1.57×
3	16.9%	31.3%–0%–1.6%	0%–42.8%–9.8%	1.25×–2.01×–1.18×

§in the order of *conv1–conv2–conv3*

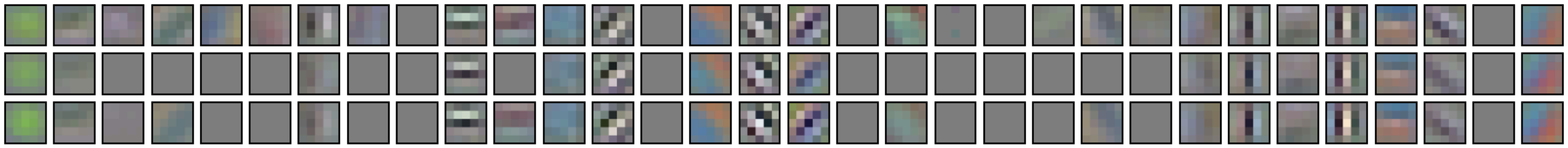


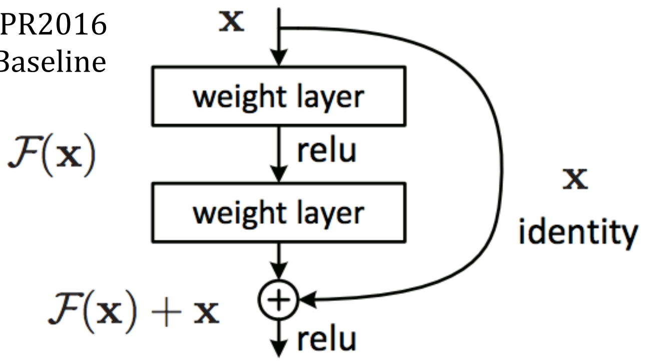
Figure 5: Learned *conv1* filters in *ConvNet 1* (top), *ConvNet 2* (middle) and *ConvNet 3* (bottom)

SSL can efficiently learn DNNs with smaller but dense weight matrix which has good locality

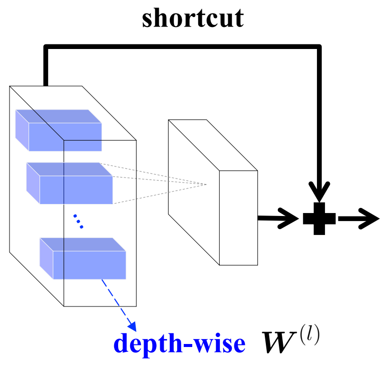
Experiments – Regularizing the depth of DNNs

Experiments of ResNets on CIFAR-10

K. He, CVPR2016
Baseline



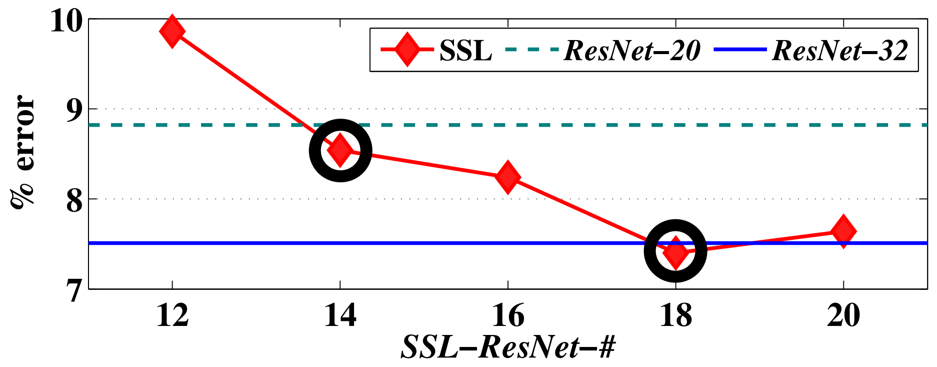
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

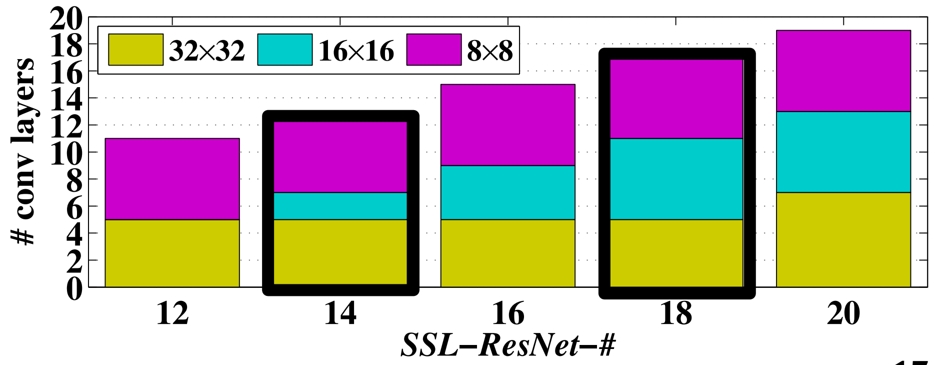


ResNet-20/32: baseline with 20/32 layers

SSL-ResNet-#: Ours with # layers after learning depth of ResNet-20

	# layers	error	# layers	error
ResNet	20	8.82%	32	7.51%
SSL-ResNet	**14**	**8.54%**	**18**	**7.40%**



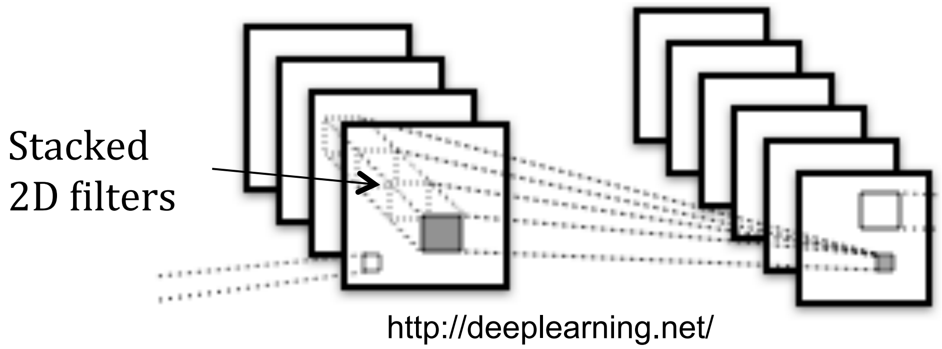


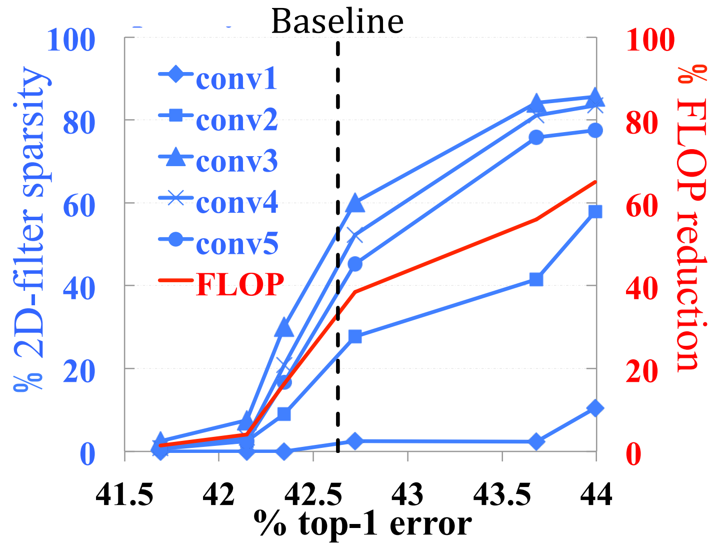
Experiments – AlexNet@ImageNet

3D convolution = sum of 2D convolutions:

$$\mathbf{F}^{(l+1)}_{c_{l+1},y_{l+1},x_{l+1}} = \sum_{c_l=1}^{C_l}\sum_{m_l=1}^{M_l}\sum_{k_l=1}^{K_l} \mathbf{F}^{(l)}_{c_l,(y_{l+1}+m_l-1),(x_{l+1}+k_l-1)} \cdot \mathbf{W}^{(l)}_{n_l,c_l,m_l,k_l}$$

Learning 2D-filter−wise sparsity

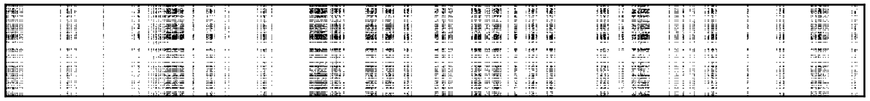




1. Save 30%–40% FLOP without accuracy loss
2. Save 60%-70% FLOP with <1.5% accuracy loss
3. Save FLOP by structurally removing 2D filters
4. Deeper layer has higher sparsity
5. Reduce the error of AlexNet by ~1%

Experiments – AlexNet@ImageNet

Learning row-wise and column–wise sparsity:



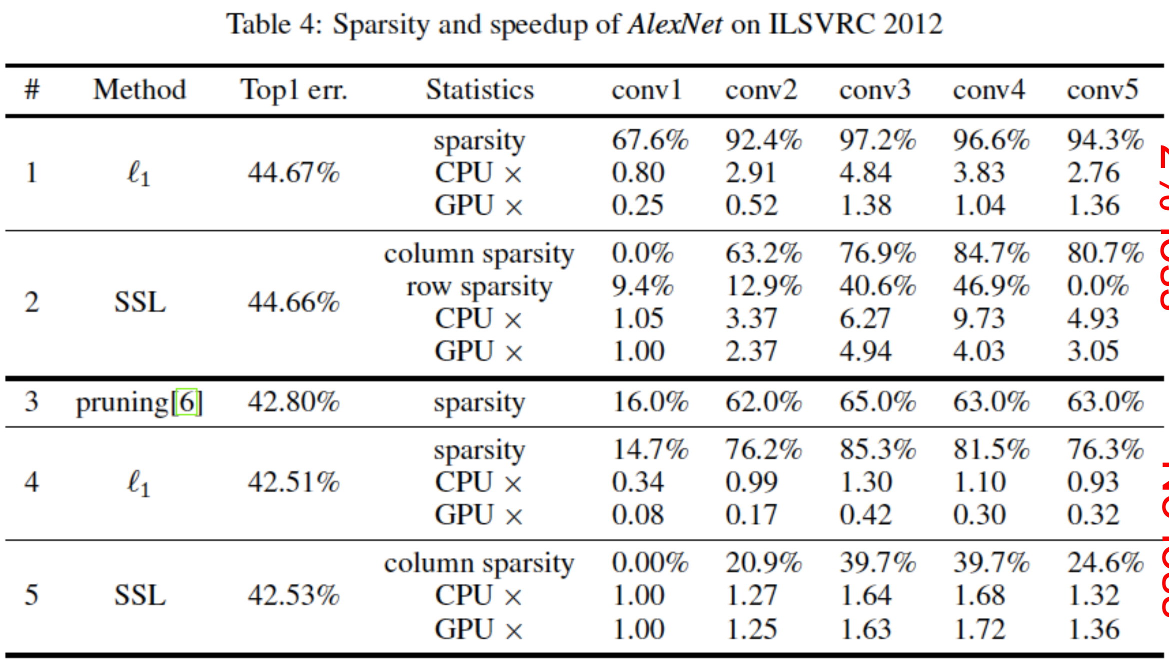
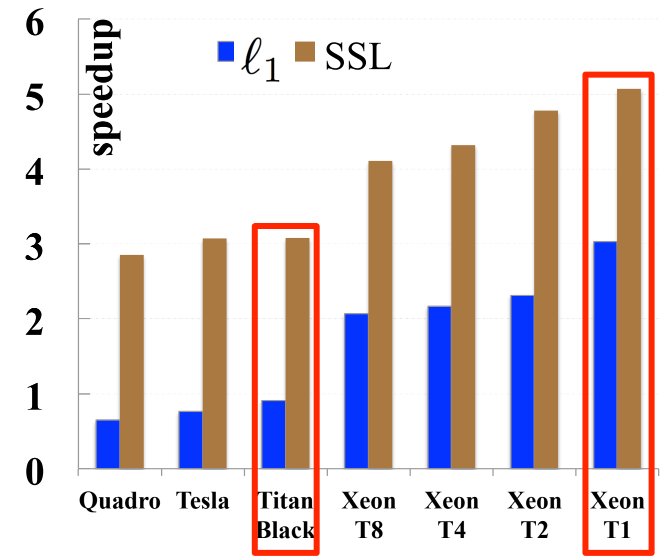


Table 4: Sparsity and speedup of *AlexNet* on ILSVRC 2012

#	Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5
1	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%
			CPU ×	0.80	2.91	4.84	3.83	2.76
			GPU ×	0.25	0.52	1.38	1.04	1.36
2	SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%
			row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%
			CPU ×	1.05	3.37	6.27	9.73	4.93
			GPU ×	1.00	2.37	4.94	4.03	3.05
3	pruning[6]	42.80%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%
4	ℓ_1	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%
			CPU ×	0.34	0.99	1.30	1.10	0.93
			GPU ×	0.08	0.17	0.42	0.30	0.32
5	SSL	42.53%	column sparsity	0.00%	20.9%	39.7%	39.7%	24.6%
			CPU ×	1.00	1.27	1.64	1.68	1.32
			GPU ×	1.00	1.25	1.63	1.72	1.36

2% loss

No loss

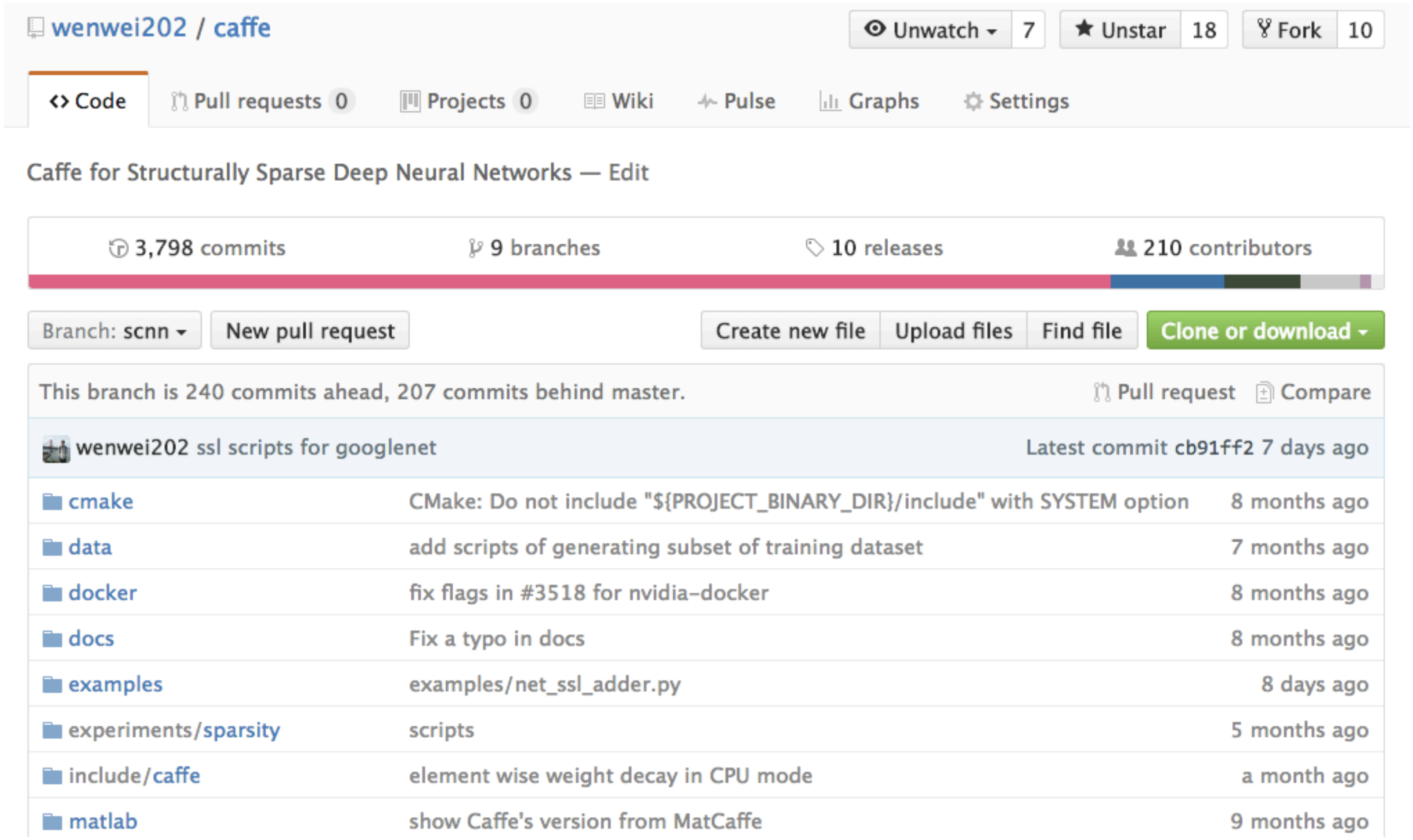


2% loss

1. Non-structured sparsity method even slows down in some layers
2. layer-wise 5.1X /3.1X on CPU/GPU with 2% accuracy loss
3. layer-wise 1.4X on both CPU and GPU w/o accuracy loss
4. Higher speedups than non-structured speedups

Open Source

Source code in Github, and trained model in model zoo


wenwei202 / caffe

Unwatch 7 | Unstar 18 | Fork 10

Code | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

Caffe for Structurally Sparse Deep Neural Networks — Edit

3,798 commits | 9 branches | 10 releases | 210 contributors

Branch: scnn | New pull request | Create new file | Upload files | Find file | Clone or download

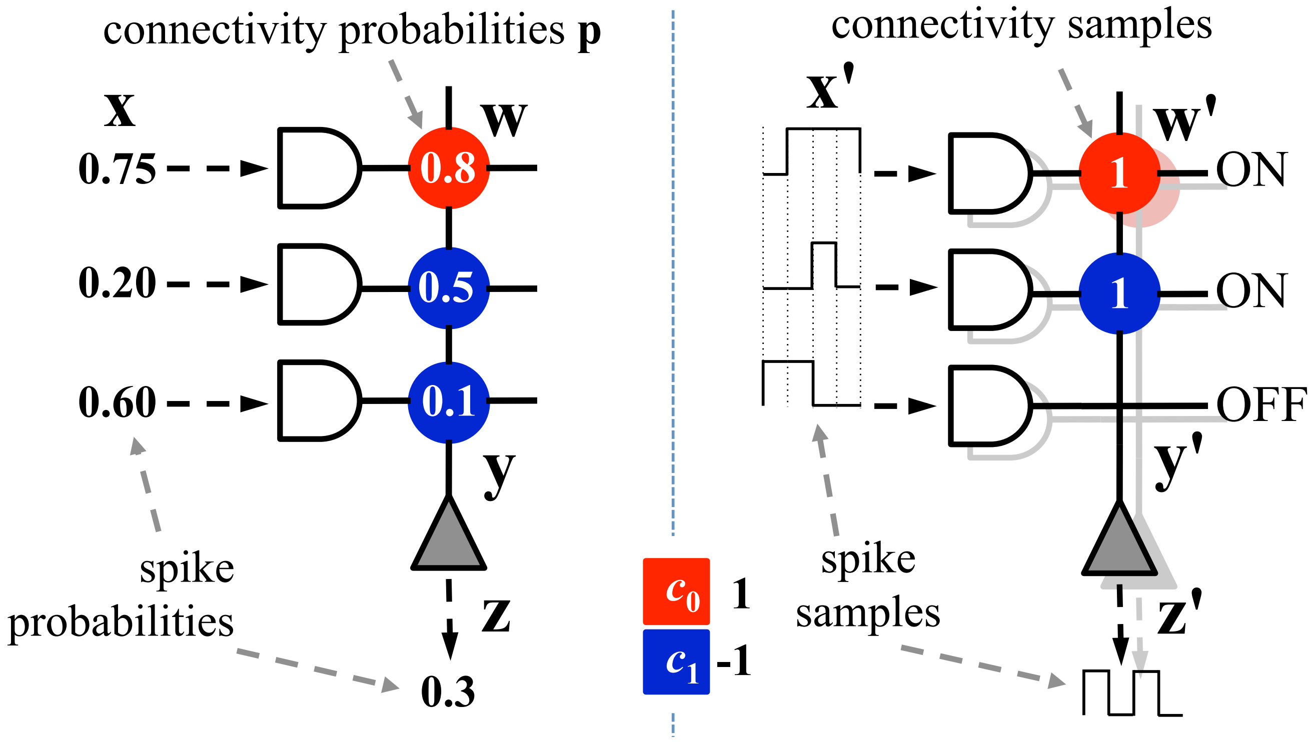
This branch is 240 commits ahead, 207 commits behind master. Pull request | Compare

wenwei202 ssl scripts for googlenet — Latest commit cb91ff2 7 days ago

cmake	CMake: Do not include "${PROJECT_BINARY_DIR}/include" with SYSTEM option	8 months ago
data	add scripts of generating subset of training dataset	7 months ago
docker	fix flags in #3518 for nvidia-docker	8 months ago
docs	Fix a typo in docs	8 months ago
examples	examples/net_ssl_adder.py	8 days ago
experiments/sparsity	scripts	5 months ago
include/caffe	element wise weight decay in CPU mode	a month ago
matlab	show Caffe's version from MatCaffe	9 months ago

https://github.com/wenwei202/caffe/tree/scnn

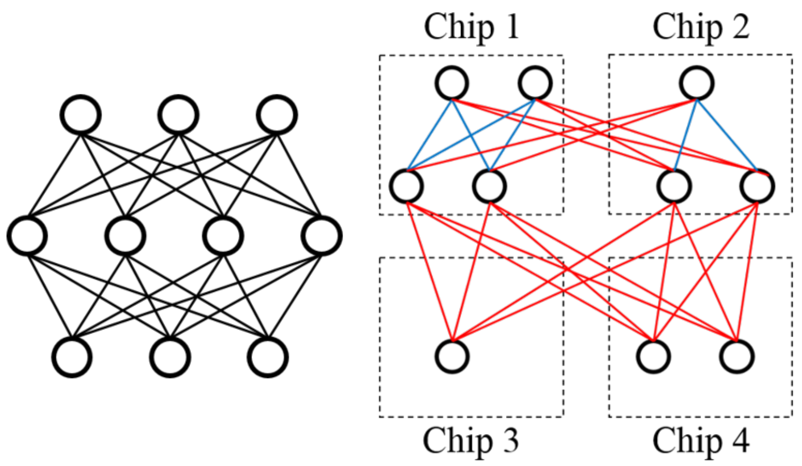
Discussion – generalizing to spiking NN



Wei Wen, *et al.*, DAC 2016
Best paper nomination

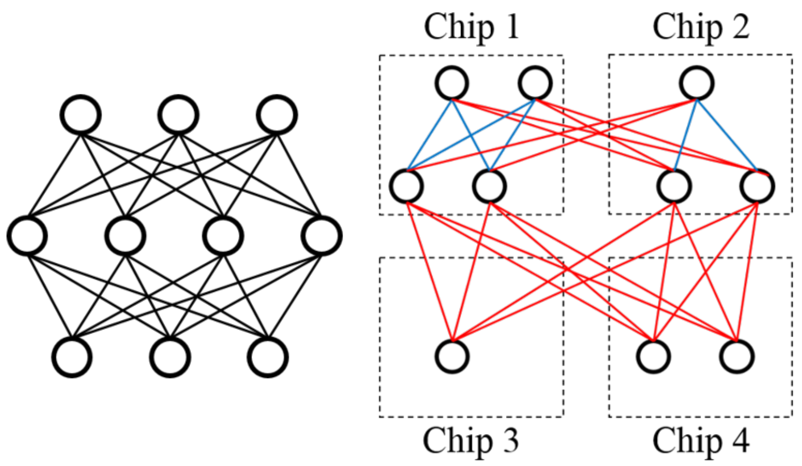
Group Lasso on weights → Group Lasso on connectivity probabilities

Discussion – generalizing to distributed systems



Reducing communication:
Split inter-chip connections to sub-groups and use SSL to zero out some sub-groups

Discussion – generalizing to distributed systems



Reducing communication: Split inter-chip connections to sub-groups and use SSL to zero out some sub-groups

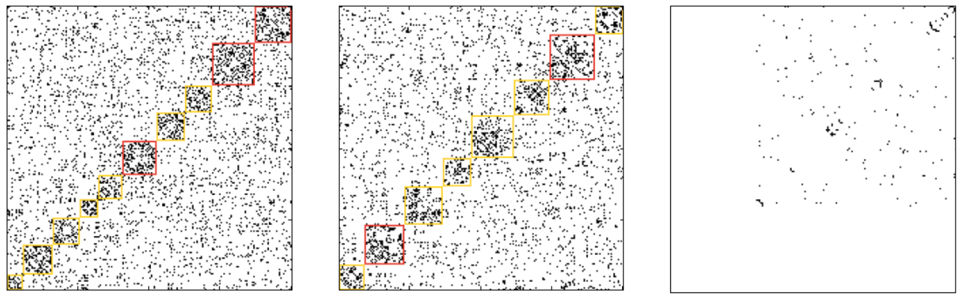


Figure 6. Results of ISC iterations.

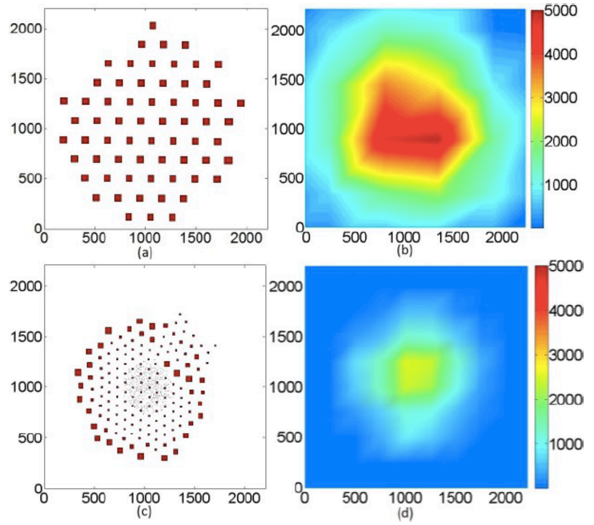


Figure 10. The placement and routing results of testbench 3 without clustering are shown in (a) and (b). The results with AutoNCS are shown in (c) and (d). Scale bar: 140 μm.

Wei Wen, *et al.*, DAC 2015
Best paper nomination

Conclusion

- We have proposed a Structured Sparsity Learning (SSL) method to learn filter, channel, filter shape, and depth structures in deep neural networks (DNNs).
- The structured sparsity in the DNN achieves significant speedups for the DNN evaluation both on CPU and GPU.
- A variant of SSL can be performed as structure regularization to improve classification accuracy of state-of-the-art DNNs.
- Our methods can generalize to spiking NNs and distributed systems
- Our method can achieve higher speedups when combining with hardware/software customization.

THANKS!