PerforatedCNNs: Acceleration through Elimination of Redundant Convolutions

Michael Figurnov, Dmitry Vetrov, Pushmeet Kohli

Skoltech
Skolkovo Institute of Science and Technology

Microsoft Research

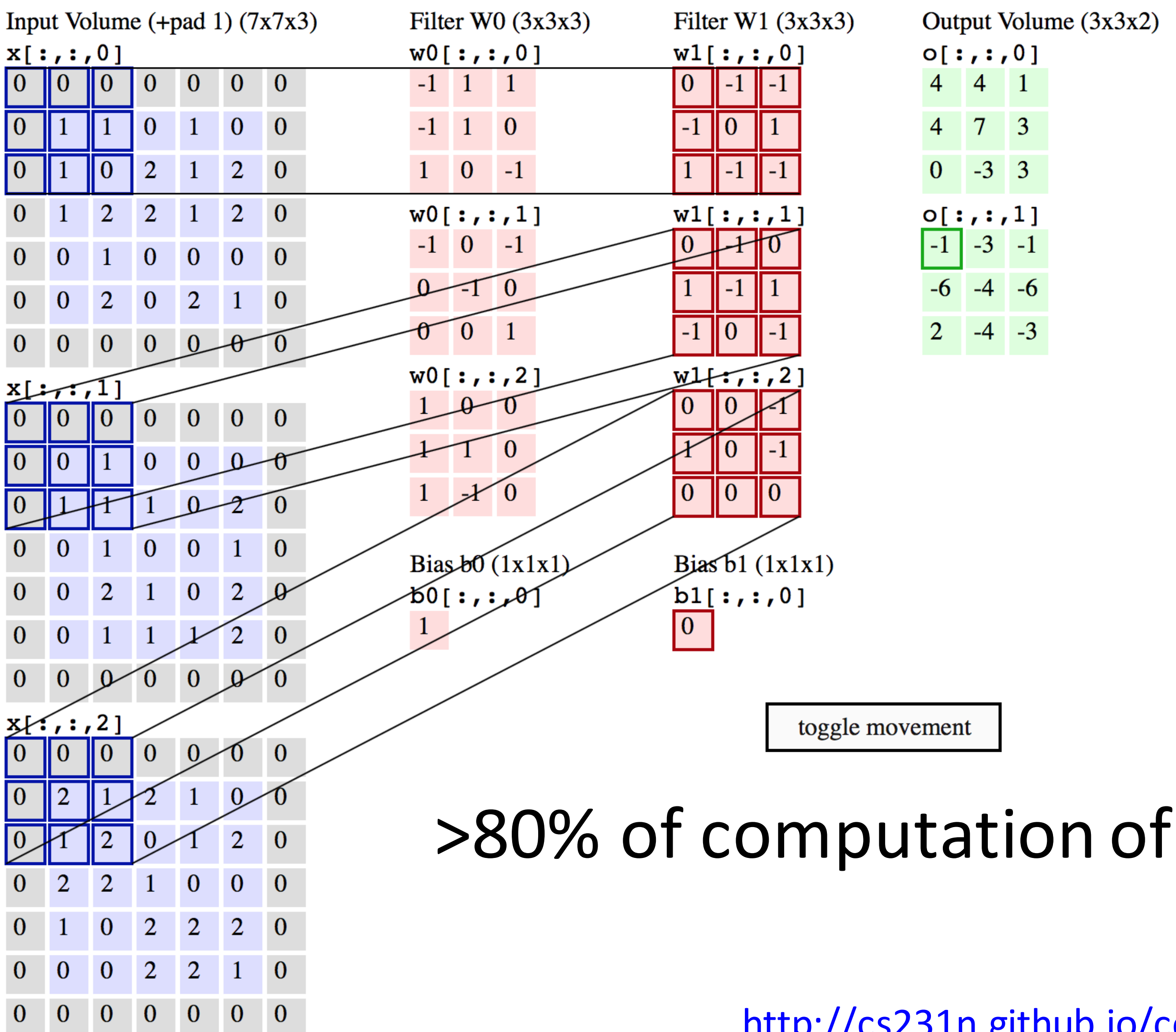
21.03.2016

VGG-16 convolutional network

- Impressive performance for vision problems (image classification, segmentation)
- 300 ms per image on a quad-core CPU
  - Too slow for real-time processing without GPU
- 15 billion multiplications per image
  - Too power-demanding for mobile devices

K. Simonyan, A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. ICLR’15

Convolutional layer



>80% of computation of CNNs!

http://cs231n.github.io/convolutional-networks/

Related work: tensor decomposition

- Decompose convolution into a sequence of convolutions with lower total complexity

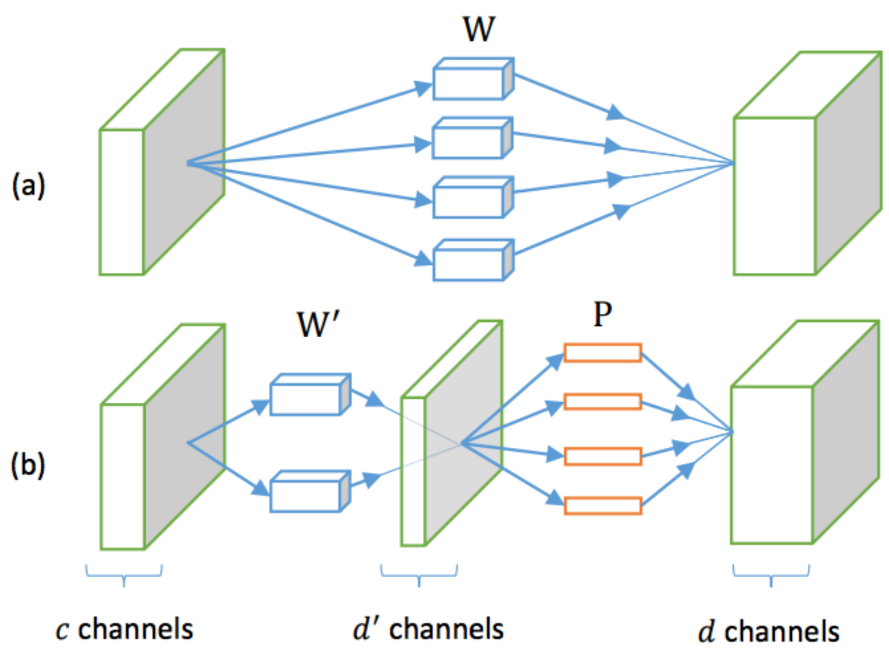


Figure 1: Illustration of the decomposition. (a) An original layer with complexity $O(dk^2c)$. (b) An approximated layer with complexity reduced to $O(d'k^2c) + O(dd')$.

X. Zhang, et al. "Accelerating Very Deep Convolutional Networks for Classification and Detection." TPAMI'15

Related work: lower precision

- Can use 16 bit floats (instead of 32 bits) with no degradation of accuracy

Gupta et al. “Deep Learning with Limited Numerical Precision.” ICML’15

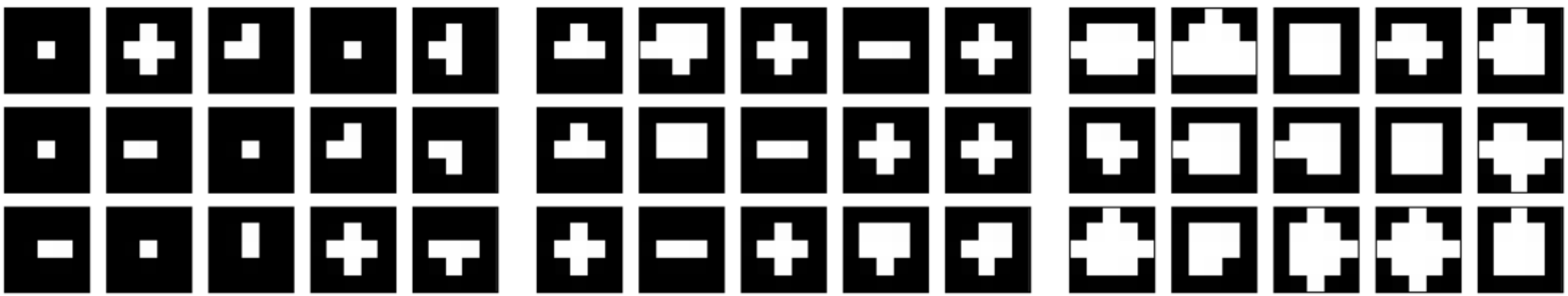
- Current area of research: **binary** connections

Courbariaux et al. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations.” NIPS’15

Rastegari et al. “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks” arxiv’16

Related work: group-wise brain damage

- Reduce the spatial size of the convolutional kernels in a smart way
- Use 3x3 kernel for some input channels, 1x1 for others



(a) sparsity $1 - \tau = 0.9$ (b) sparsity $1 - \tau = 0.8$ (c) sparsity $1 - \tau = 0.6$

Figure 4: The sparsity patterns obtained by group-wise brain damage on the second convolutional layer of AlexNet for different sparsity levels. Nonzero weights are shown in white. In general, group-wise brain damage shrinks the receptive fields towards the center and tends to make them circular.

V. Lebedev, V. Lempitsky. "Fast convnets using group-wise brain damage." arXiv'15

Loop perforation

```
float sum = 0;
for (int i = 0; i < N; i++) {
  sum += a[i];
}
float mean = sum / N;
```

```
float sum = 0;
for (int i = 0; i < N; i += 2) {
  sum += a[i];
}
float mean = sum / (N/2);
```

Trading accuracy for speed

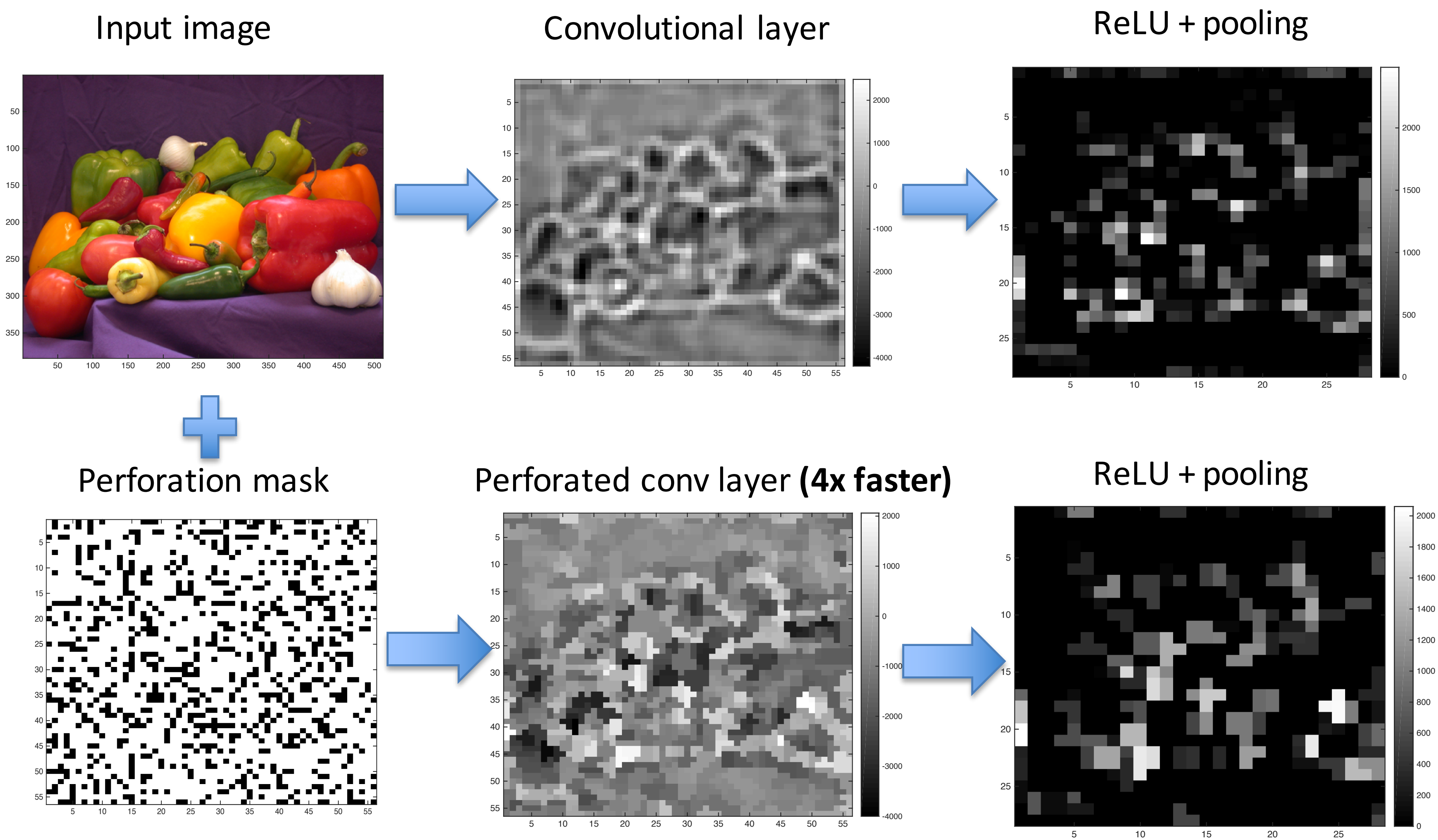
S. Misailovic, D.M. Roy, and M.C. Rinard. Probabilistically accurate program transformations. In Static Analysis, pages 316–333. Springer, 2011

S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, pages 25–34. ACM, 2010

Perforated convolutional layer

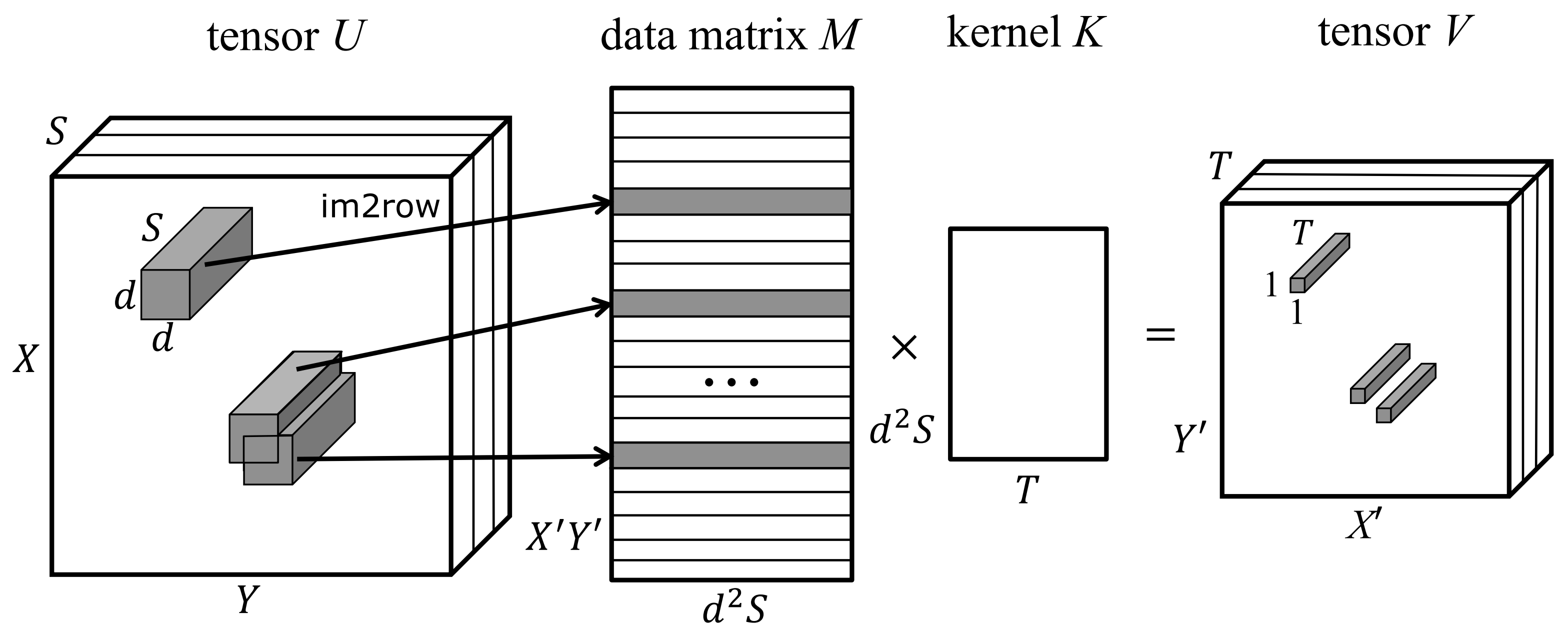
- Goals:
  - Small decrease of the network’s accuracy
  - Possibility of efficient implementation
- Outputs of convolutional layers are spatially redundant
- Perforated convolutional layer:
  - Calculate the outputs a convolutional layer in a subset of spatial positions
  - Interpolate the missing values using nearest neighbor
- Why does this work?
  - ReLU and max-pooling ignore most values in the network

Perforated convolutional layer



Efficient implementation

"Caffe-style" convolution: reduction to matrix multiplication



Perforation = skipping rows of data matrix M

Interpolation is performed implicitly in the next layer's im2row

Pros & cons

\+ Less computation: smaller data matrix

\+ Efficient: 50-100% of theoretical speedup

\+ Less memory: fewer activations to store

\+ Works well with subsequent 1x1 convolutions

\+ Does not change architecture of the network

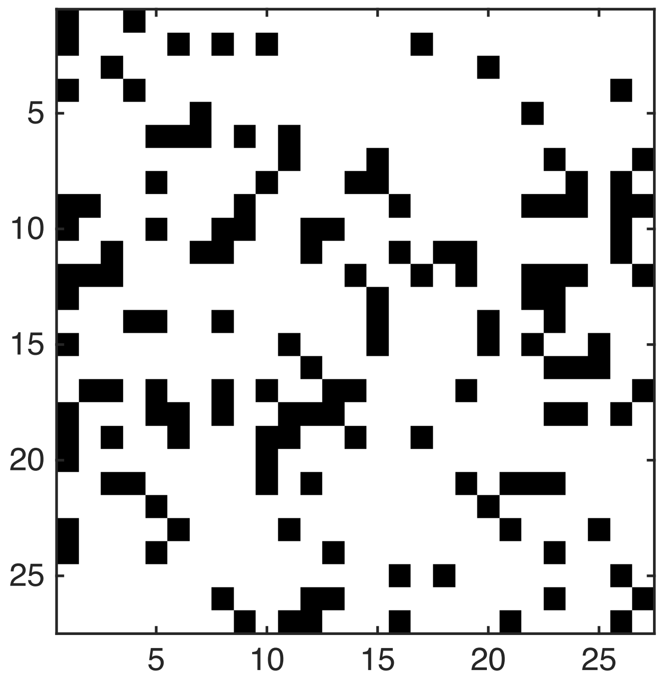
\+ *Mask can be dynamically adjusted* – future work

\- Requires custom implementation

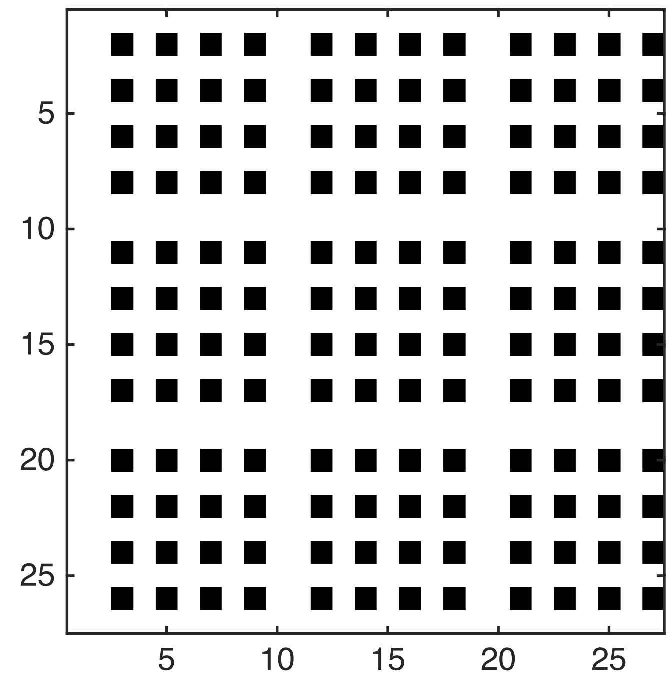
\- Need to choose the **perforation masks**

Baseline perforation masks

Uniform



Grid

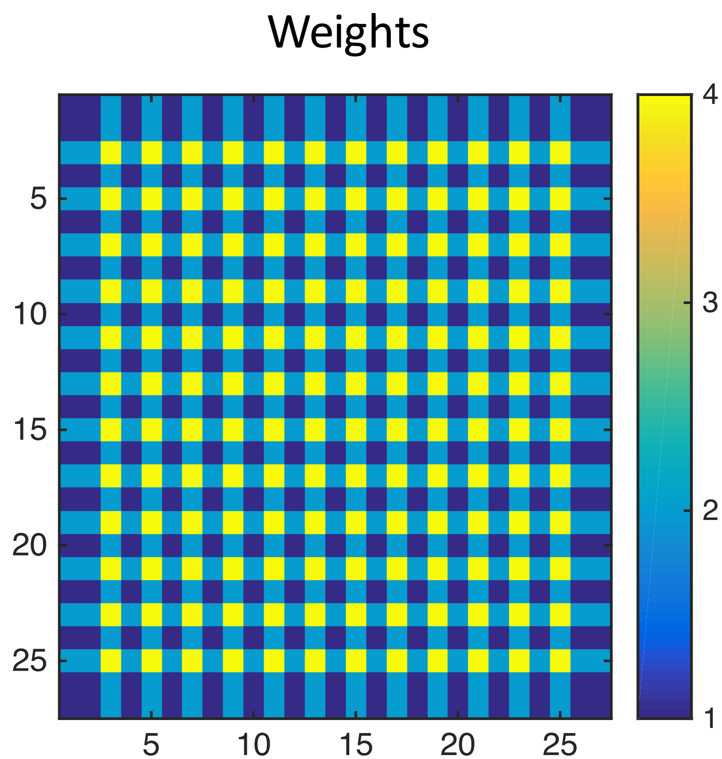


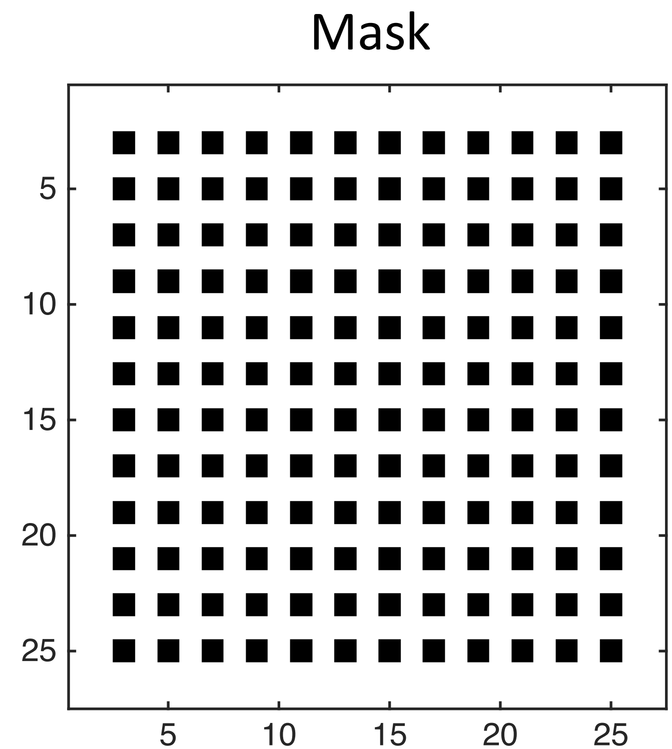
Similar to increasing the stride of convolution

Pooling structure perforation mask

Weight is the number of times the position is used in the next pooling layer

AlexNet conv2: followed by 3x3 pooling with stride 2





Output positions are not equally important!

How can we measure their impact?

Impact perforation mask

- Estimate relative importance of spatial positions for the loss (possibly for a perforated network!)
- First-order Taylor expansion:

L(V) – loss as a function of outputs of convolutional layer *V*

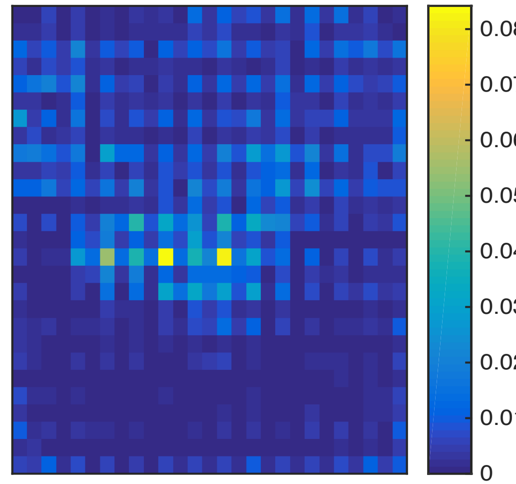
V' is V with position (x_0, y_0, t_0) replaced with zero

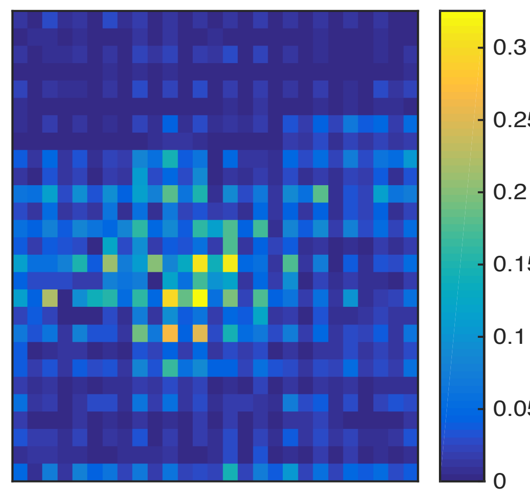
$$
\begin{aligned}
|L(V') - L(V)| &\approx \Big| \sum_{x=1}^{X} \sum_{y=1}^{Y} \sum_{t=1}^{T} \frac{\partial L(V)}{\partial V(x,y,t)} (V'(x,y,t) - V(x,y,t)) \Big| \\
&= \Big| \frac{\partial L(V)}{\partial V(x_0,y_0,t_0)} V(x_0,y_0,t_0) \Big|.
\end{aligned}
$$

Aggregate over channels:

$$
G(x,y;V) = \sum_{t=1}^{T} \Big| \frac{\partial L(V)}{\partial V(x,y,t)} V(x,y,t) \Big|
$$

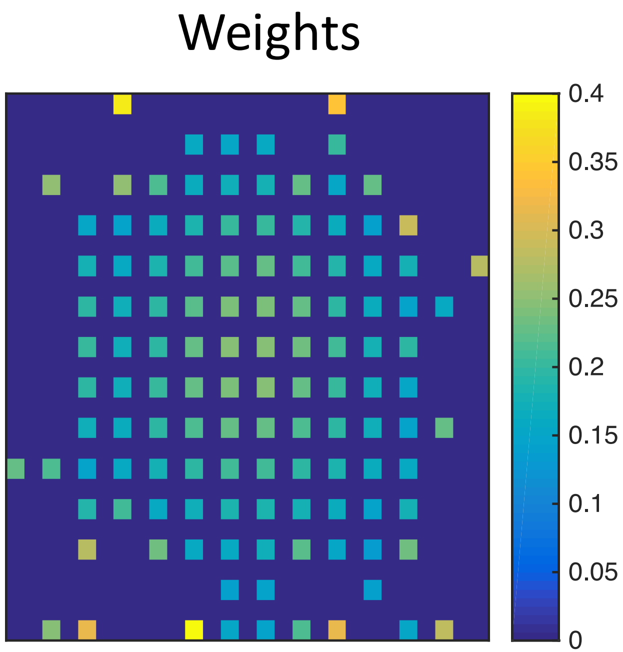
Per-image impacts G(x, y; V) for AlexNet conv2

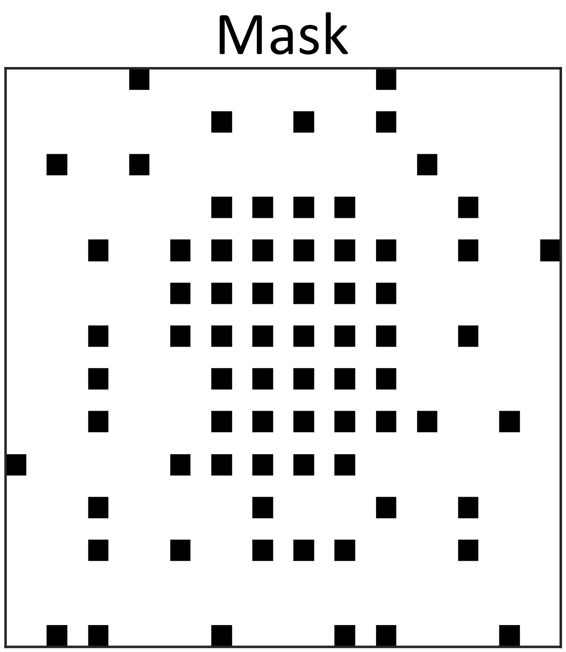




Impact perforation mask

- After averaging impacts over the training dataset (for an already perforated network)

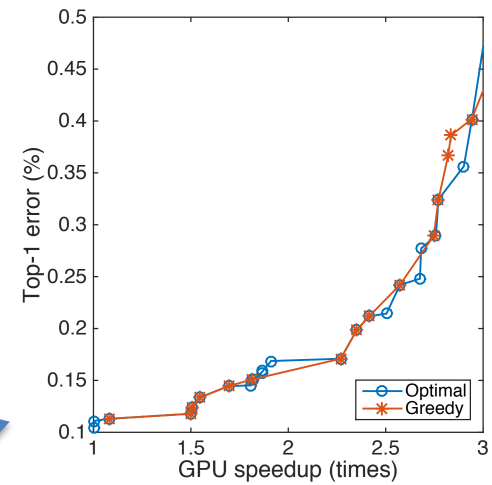




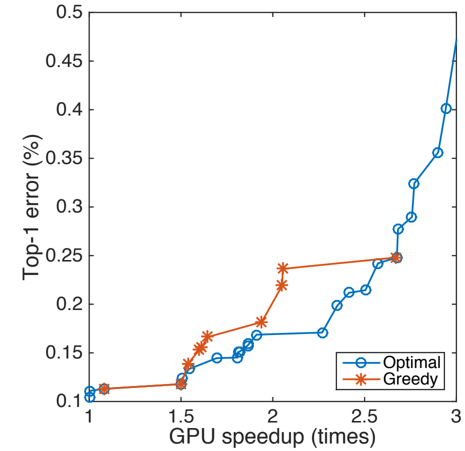
- Already-perforated positions have zero weight
- Iterate between increasing perforation and recalculating weights

Perforating multiple layers

- Greedy algorithm
- *NLL* is class negative log-likelihood, *t* is network evaluation time
- Iteratively perforate the layer with the minimal value of the cost function $\frac{NLL_n - NLL_0}{t_0 - t_n}$



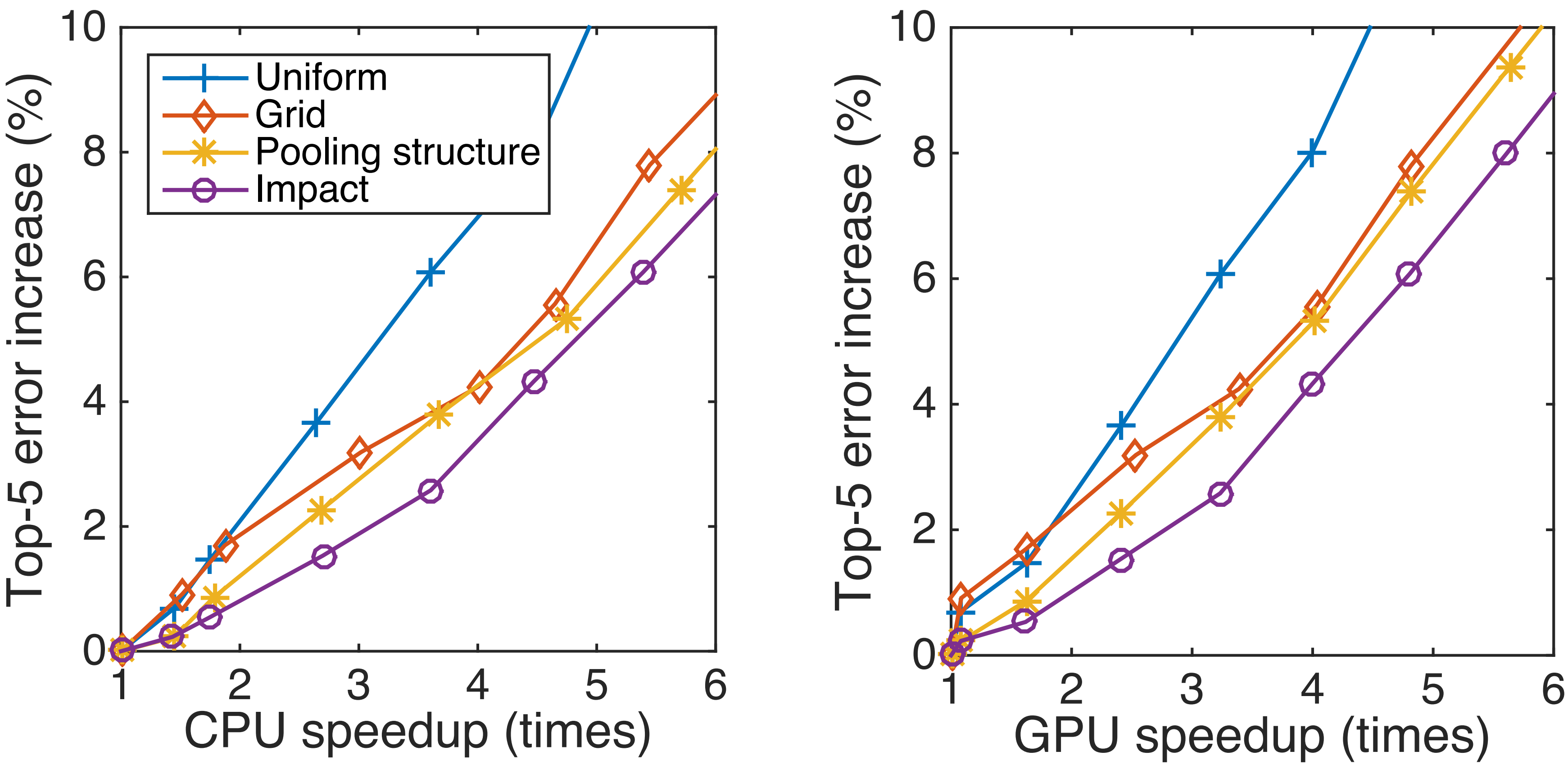
- Surprisingly, this cost function is much better than $\frac{NLL_n - NLL_{n-1}}{t_{n-1} - t_n}$



Experiments

What is the best perforation mask?

Conv2 layer of AlexNet, no fine-tuning



Comparison with state-of-the-art

Conv2 layer of AlexNet, after fine-tuning

Method	CPU time ↓	Error ↑ (%)
Impact mask, $r = \frac{3}{4}$, 3×3 filters	**9.1×**	+1
Impact mask, $r = \frac{5}{6}$	5.3×	+1.4
Impact mask, $r = \frac{4}{5}$	4.2×	+0.9
(Lebedev & Lempitsky, 2015)	**10×**	top-1 +1.1
(Lebedev & Lempitsky, 2015)	5×	top-1 +0.4
(Jaderberg et al., 2014)	6.6×	+1
(Lebedev et al., 2015)	4.5×	+1
(Denton et al., 2014)	2.7×	+1

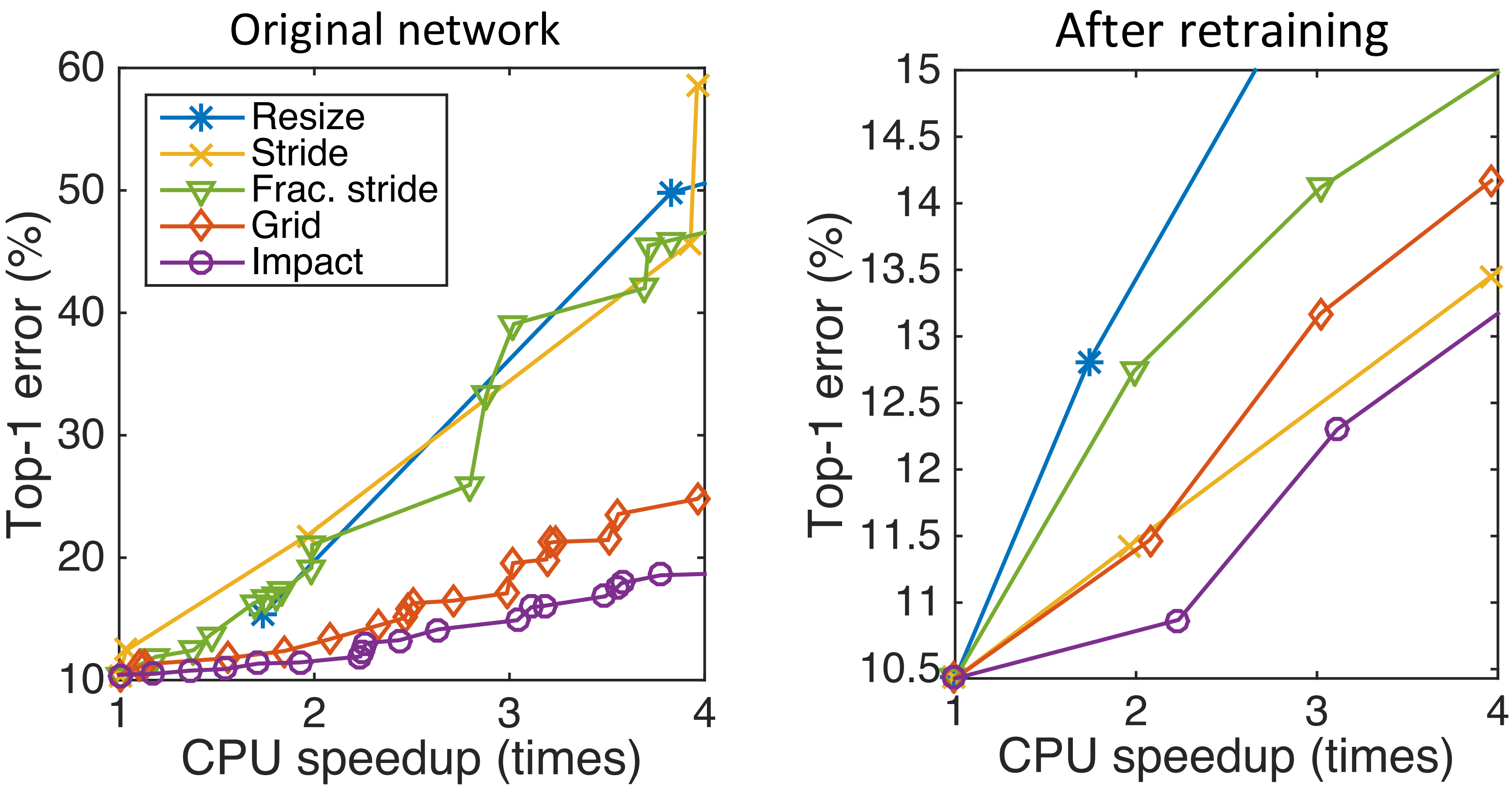
V. Lebedev, V. Lempitsky. "Fast convnets using group-wise brain damage." arXiv'15
M. Jaderberg, A. Vedaldi, A. Zisserman. "Speeding up convolutional neural networks with low rank expansions." BMVC'14
V. Lebedev, et al. "Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition." ICLR'15
E. Denton, et al. "Exploiting linear structure within convolutional networks for efficient evaluation." NIPS'14

Baseline strategies (CIFAR10 NIN)

Resize: smaller input image

(Frac.) Stride: increase stride of convolutions

Grid & Impact: perforation

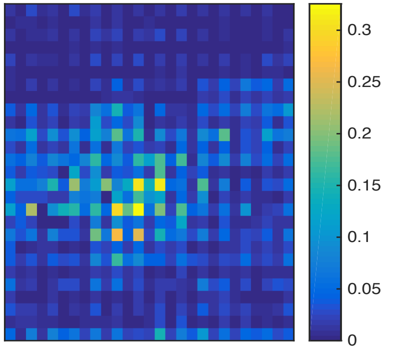


ImageNet networks results

Network	Device	Speedup	Mult. $\downarrow$	Mem. $\downarrow$	Error $\uparrow$ (%)	Tuned error $\uparrow$ (%)
AlexNet	CPU	2.0×	2.1×	1.8×	+10.7	+2.3
		3.0×	3.5×	2.6×	+28.0	+6.1
		3.6×	4.4×	2.9×	+60.7	+9.9
	GPU	2.0×	2.0×	1.7×	+8.5	+2.0
		3.0×	2.6×	2.0×	+16.4	+3.2
		4.1×	3.4×	2.4×	+28.1	+6.2
VGG-16	CPU	2.0×	1.8×	1.5×	+15.6	+1.1
		3.0×	2.9×	1.8×	+54.3	+3.7
		4.0×	4.0×	2.5×	+71.6	+5.5
	GPU	2.0×	1.9×	1.7×	+23.1	+2.5
		3.0×	2.8×	2.4×	+65.0	+6.8
		4.0×	4.7×	3.4×	+76.5	+7.3

Future work

- Data-dependent perforation masks
  - Hard attention
  - Tricky to tune



- Perforation + elimination of cross-channel redundancy

X. Zhang, et al. "Accelerating Very Deep Convolutional Networks for Classification and Detection.” TPAMI’15

Conclusion

- Modern convolutional networks are redundant
- PerforatedCNNs exploit spatial redundancy to decrease the computational cost and the memory consumption
  - 2x faster VGG-16, 1.7x less memory, 1.1% increase of top-5 err
- Architecture of the network is not changed
  - Same parameters, same intermediate activations
  - Easy to combine with other acceleration methods

Questions?

More details: ICLR’16 workshop paper
http://arxiv.org/abs/1504.08362

Code
https://github.com/mfigurnov/perforated-cnn-matconvnet
https://github.com/mfigurnov/perforated-cnn-caffe