Compressing DMA Engine: Leveraging Activation Sparsity For Training Deep Neural Networks

Minsoo Rhu [†], Mike O'Connor[*], Niladrish Chatterjee[*],
Jeff Pool[*], Youngeun Kwon [†], and Stephen W. Keckler[*]

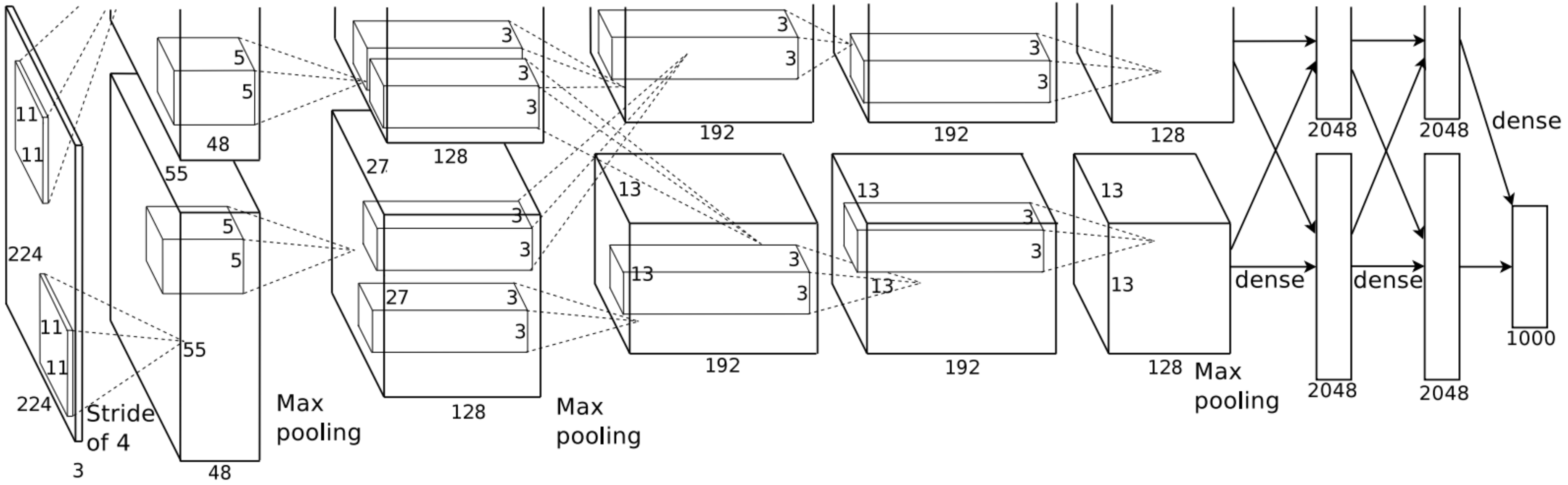
POSTECH [†] and NVIDIA[*]

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Motivation

ML trends: deeper & larger DNN models

From AlexNet to ResNet



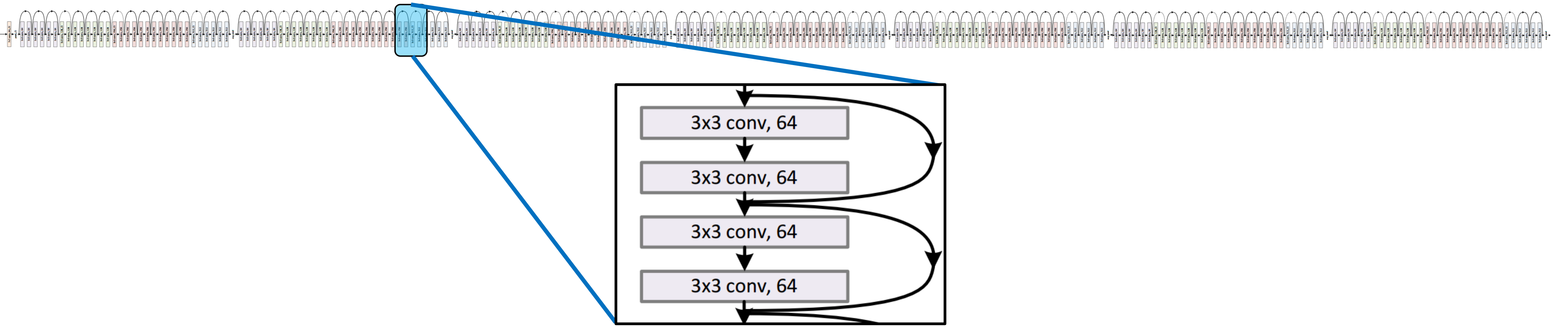
7 convolutional layers
(2012)

[AlexNet*]

* Krizhevsky et al., "ImageNet Classification with Deep Convolutional Neural Networks", NIPS-2012

ML trends: deeper & larger DNN models

From AlexNet to ResNet



153 convolutional layers
(2016)

[ResNet*]

* He et al., "Deep Residual Learning for Image Recognition", CVPR-2016

Memory "capacity" limits in DNN training

Training large & deep DNNs incurs large memory allocations

Medium

Synced Follow
In-Depth AI Technology & Industry Review www.syncedreview.com | www.jiqizhixin.com
Apr 29 · 7 min read

How to Train a Very Large and Deep Model on One GPU?

Problem: GPU memory limitation

CS231n Convolutional Neural Networks for Visual Recognition

Computational Considerations

The largest bottleneck to be aware of when constructing ConvNet architectures is the memory bottleneck. Many modern GPUs have a limit of 3/4/6GB memory, with the best GPUs having about 12GB of memory. There are three major sources of memory to keep track of:

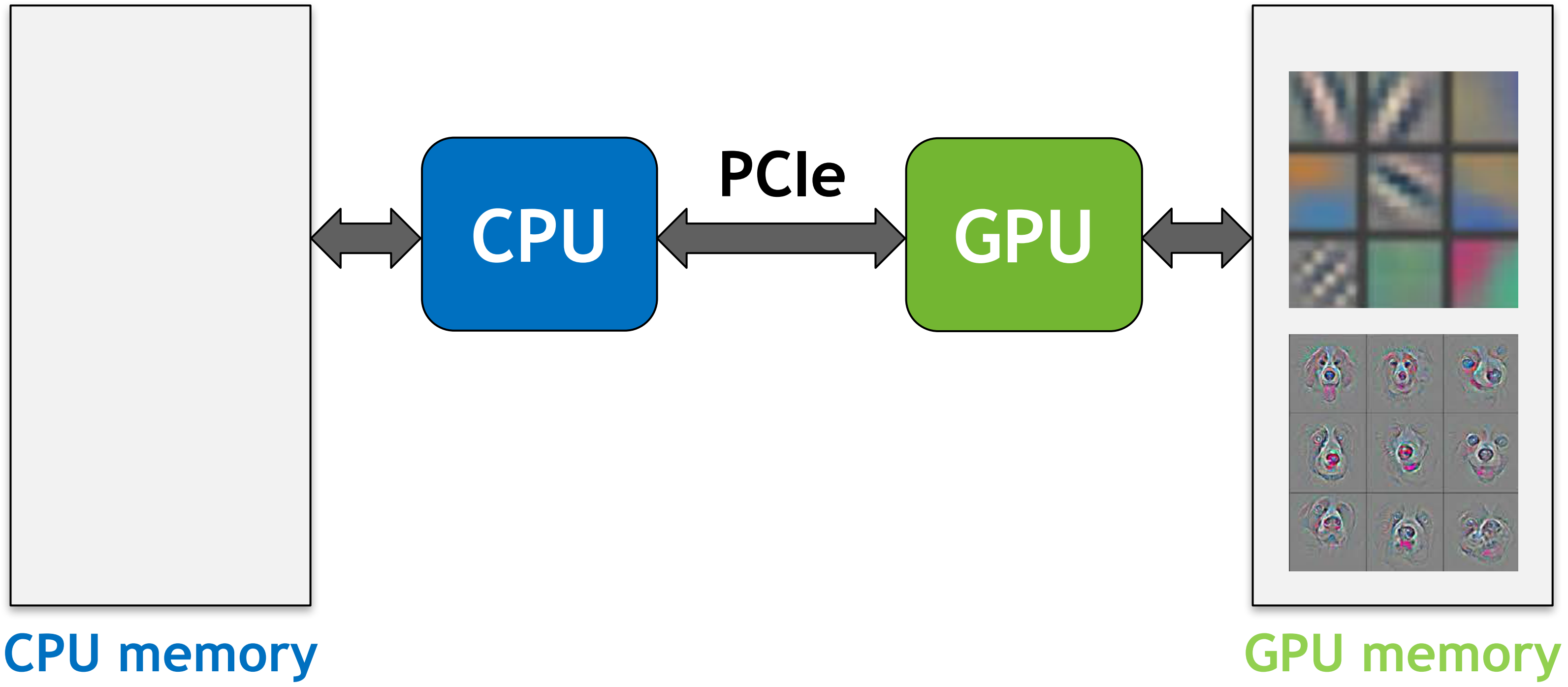
TOPBOTS POPULAR BUSINESS TECHNO

HOW TO SOLVE THE MEMORY CHALLENGES OF DEEP NEURAL NETWORKS

Posted by Jamie Hanlon | Mar 30, 2017

Prior solution: virtualized DNN (vDNN)

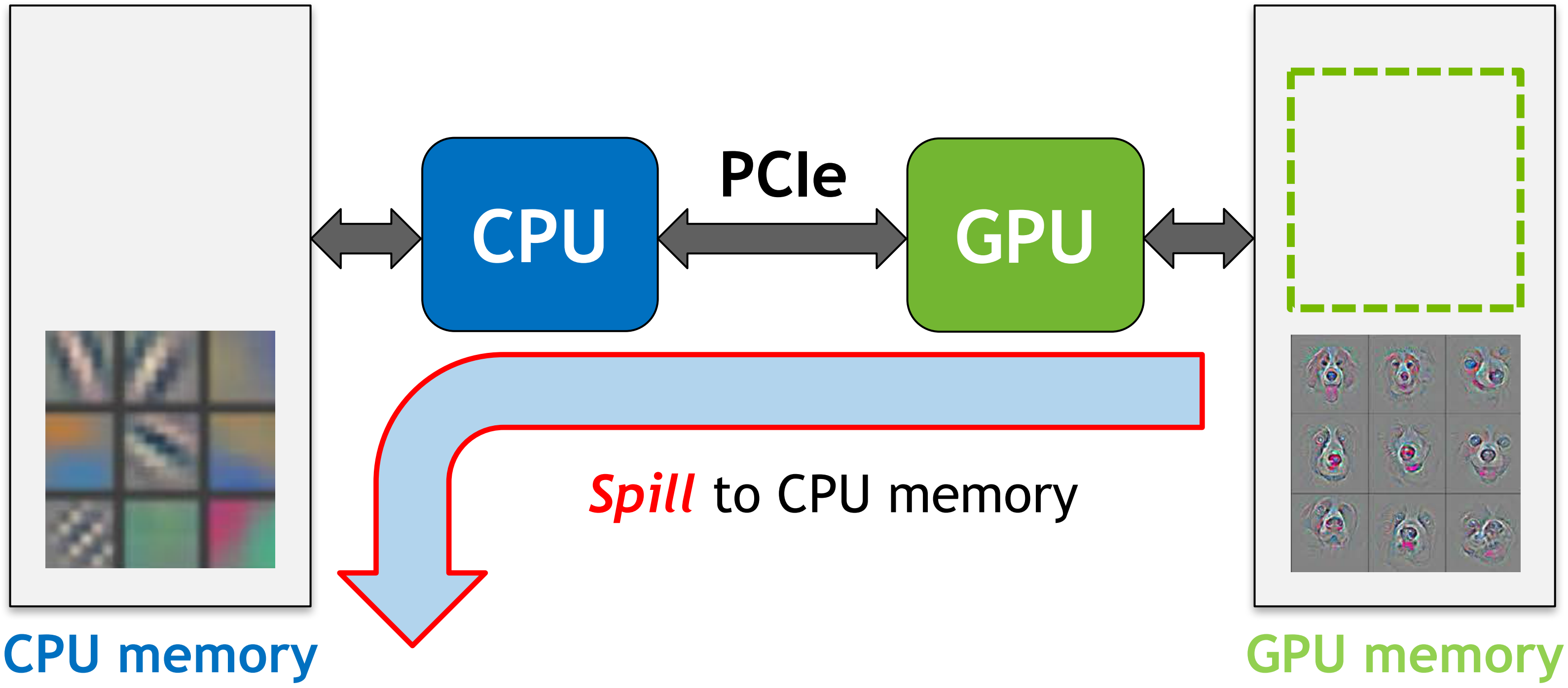
Expose both CPU and GPU memory for allocating DNN training data



* Rhu et al.,"vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design", MICRO-2016

Prior solution: virtualized DNN (vDNN)

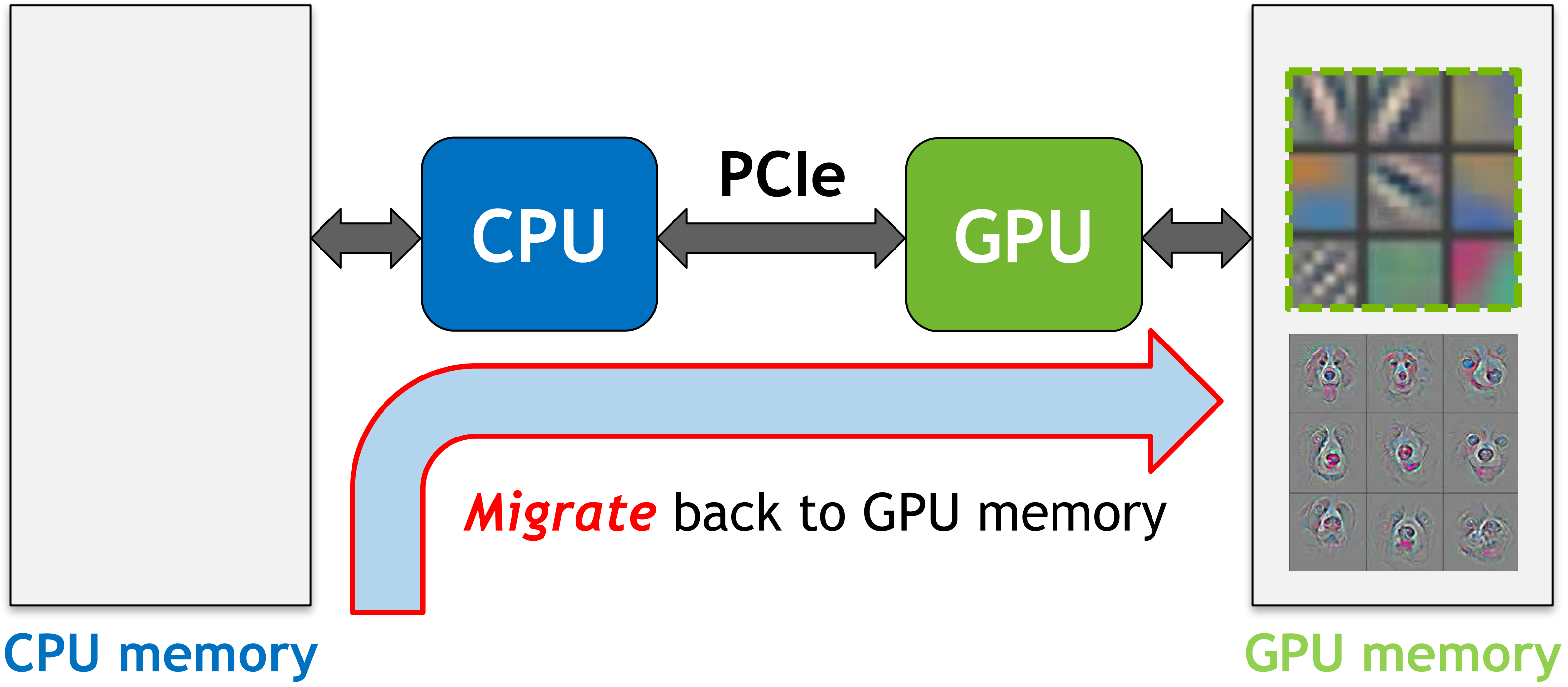
Expose both CPU and GPU memory for allocating DNN training data



* Rhu et al.,"vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design", MICRO-2016

Prior solution: virtualized DNN (vDNN)

Expose both CPU and GPU memory for allocating DNN training data



* Rhu et al.,"vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design", MICRO-2016

Large Model Support (LMS) with PowerAI

Expose both CPU and GPU memory for allocating DNN training data

Realizing the value of Large Model Support (LMS) with PowerAI IBM Caffe

SarithaVinod
Published on September 22, 2017

IBM PowerAI 4.0 has been released with Large Model Support (LMS) in IBM Caffe. LMS uses system memory in conjunction with GPU memory to overcome GPU memory limitations in Deep Learning Training.

LMS enables processing of high definition images, large models, and higher batch sizes that doesn't fit in GPU memory today (Maximum GPU memory available in Nvidia P100 GPUs is 16GB).

LMS Options

- \- lms <size in KB>>
- \- lms_frac <x>, where 0<x<1.0

You can enable the large model support in IBM Caffe by adding -lms <size in KB>>. This acts as a threshold size that decides which memory allocations will happen on CPU memory or on GPU memory.

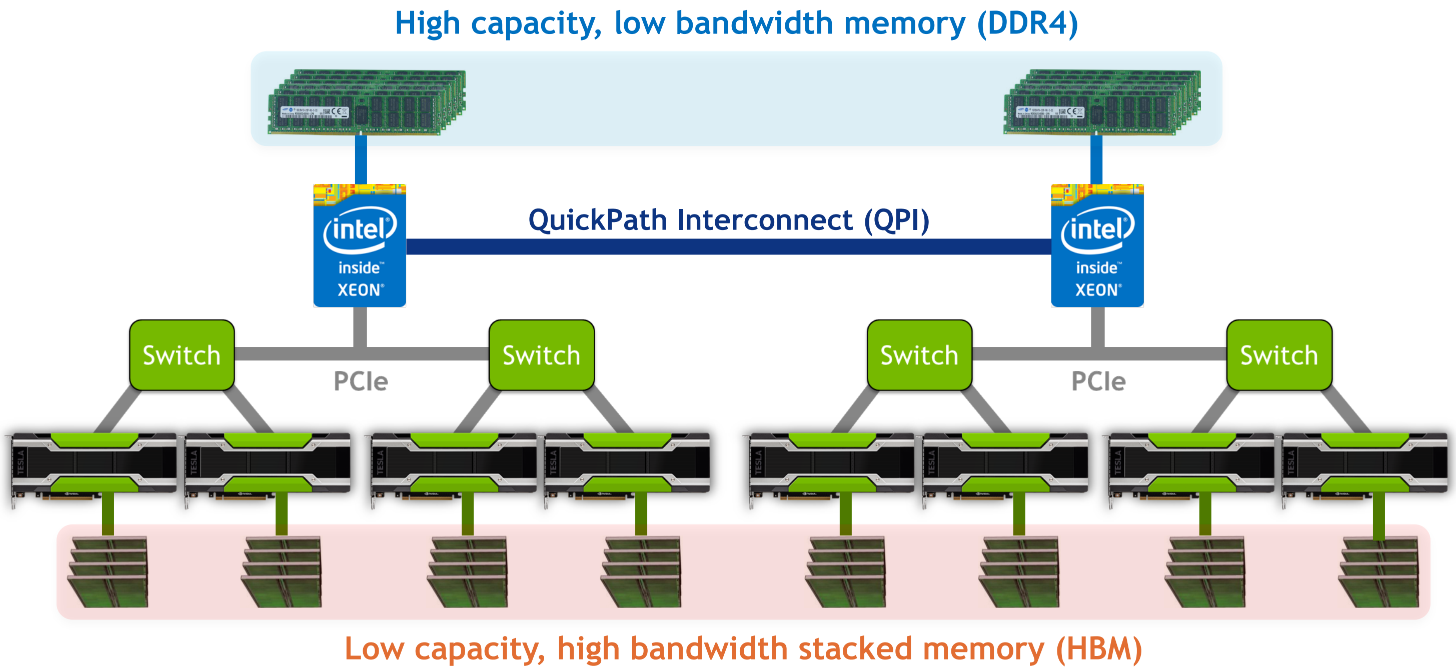
For example -lms 1000. With this option, any memory chunk allocation larger than 1000KB will be done in CPU memory, and fetched to GPU memory only when needed for computation. Thus, if you use a very large value like -lms 10000000000, it will effectively disable the feature while a small value means a more aggressive LMS. The value is used to control the performance trade-off. Apparently bringing in more data from the CPU memory will incur as overhead in runtime.

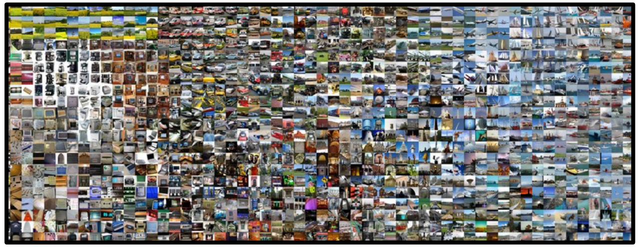
As a secondary option, there is -lms_frac <x>, where 0<x<1.0. For example, with -lms_frac 0.5 LMS doesn't kick in until more than at least 50% of GPU memory is expected to be utilized. This is useful for disabling LMS for a small network or to use the GPU memory efficiently for larger networks.

* https://developer.ibm.com/linuxonpower/2017/09/22/realizing-value-large-model-support-lms-powerai-ibm-caffe/

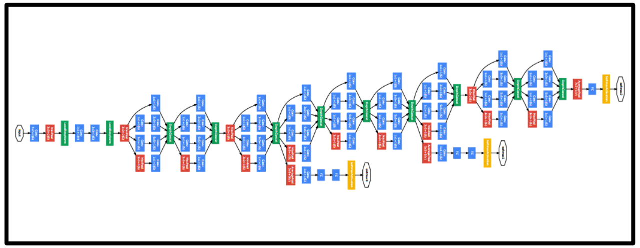
HPC system node for deep learning

Multiple GPUs (4 to 8) connected under a PCIe root complex





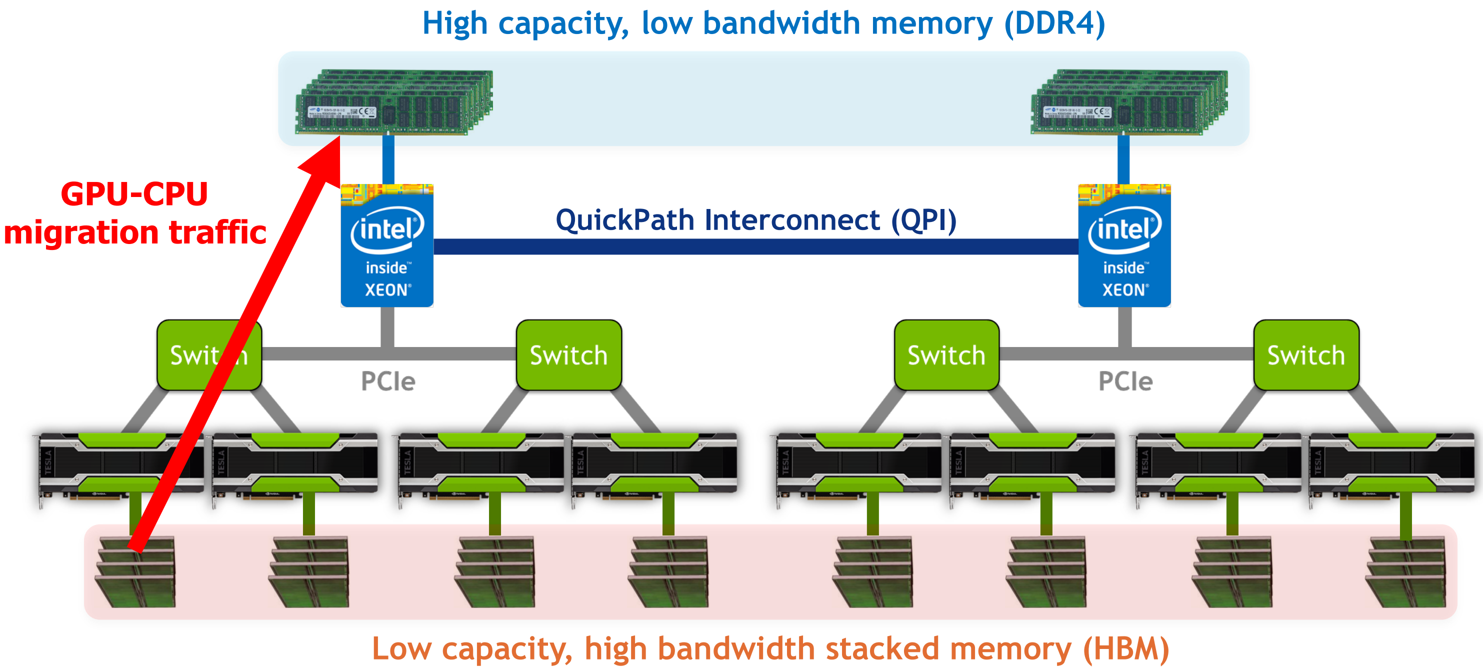
Big Data

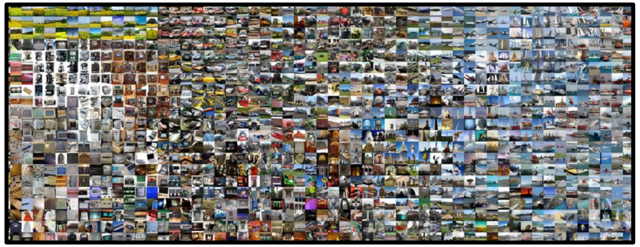


Deeper & wider neural networks

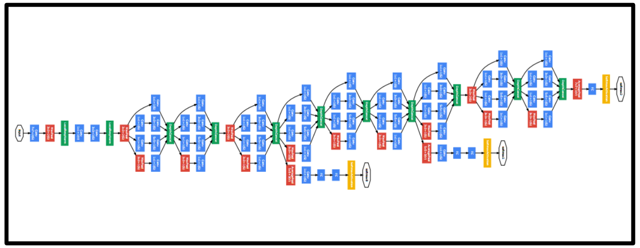
HPC system node for deep learning

Multiple GPUs (4 to 8) connected under a PCIe root complex





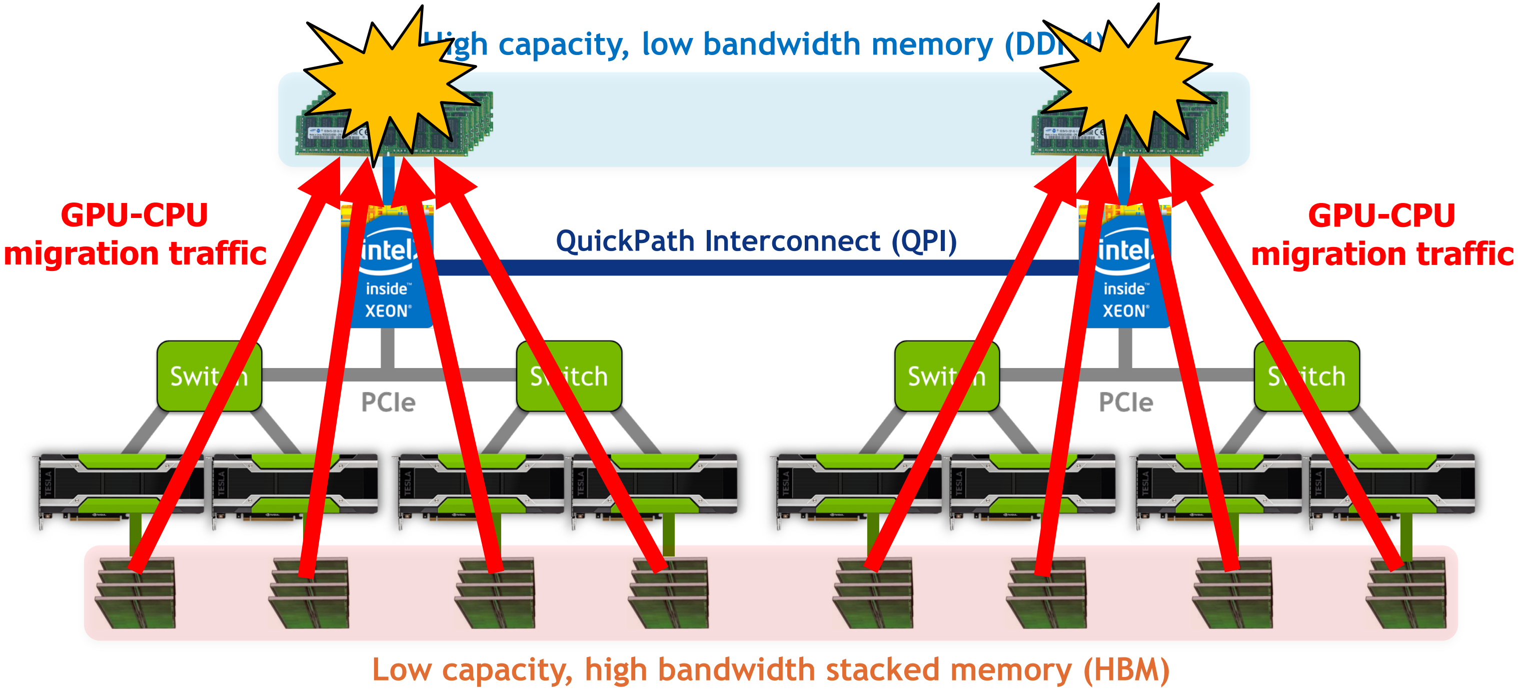
Big Data

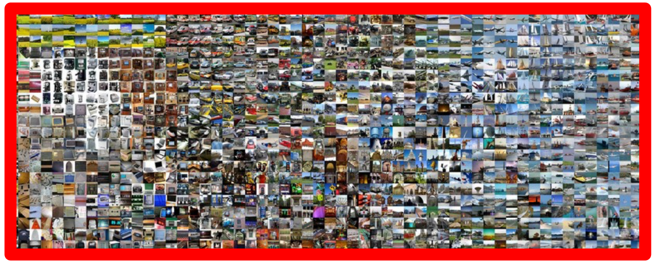


Deeper & wider neural networks

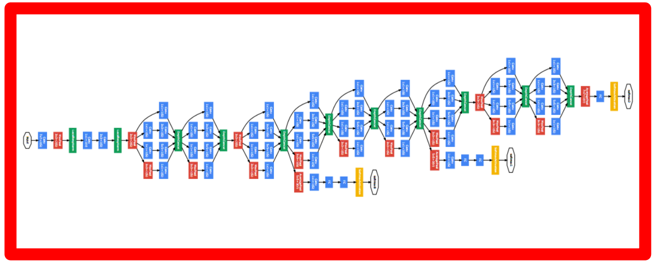
HPC system node for deep learning

Multiple GPUs (4 to 8) connected under a PCIe root complex





Big Data

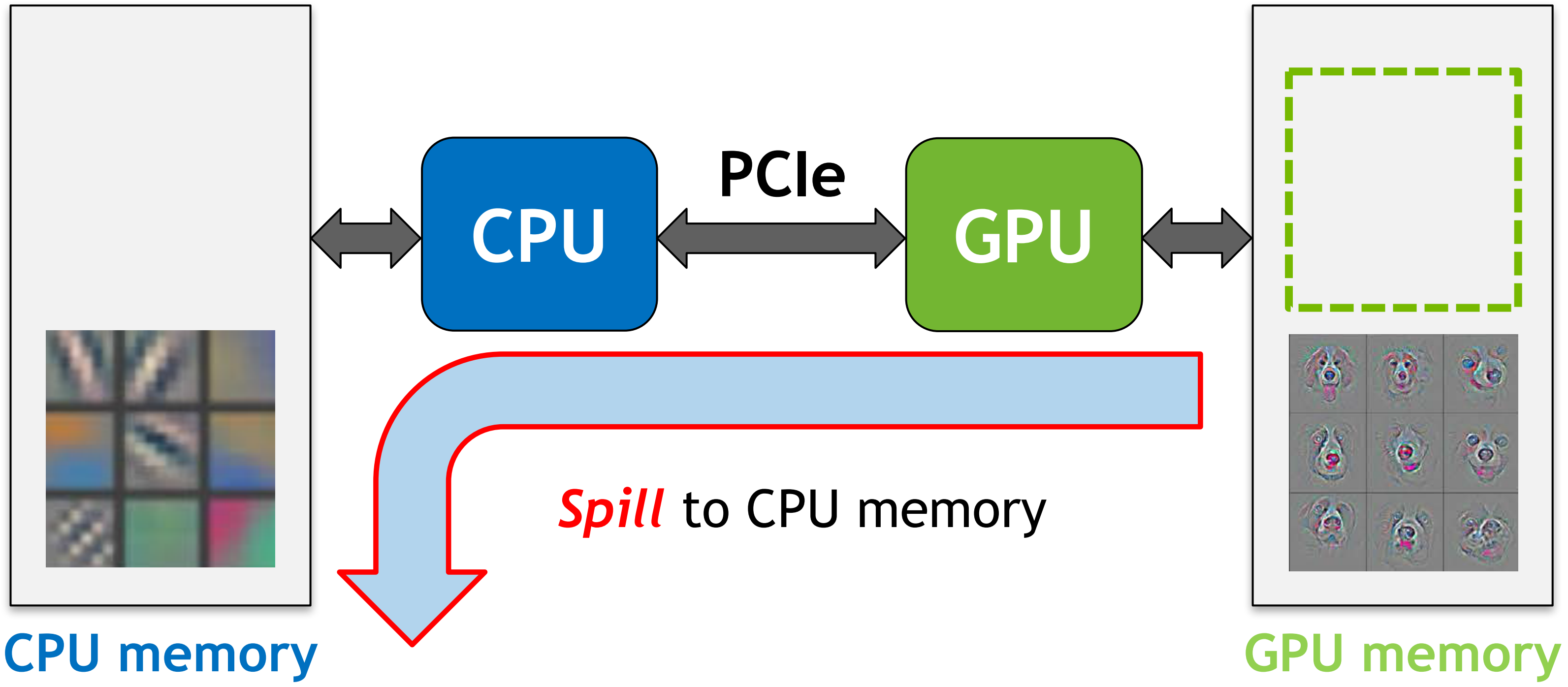


Deeper & wider neural networks

Challenges: PCIe channel bandwidth becomes a performance bottleneck!

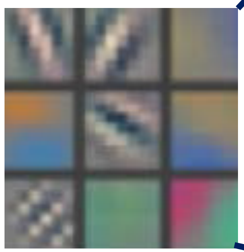
Opportunity: “sparse” data structures

Amplify *effective* PCIe bandwidth via compressing CPU-migrated data



Opportunity: "sparse" data structures

Amplify *effective* PCIe bandwidth via compressing CPU-migrated data



0	0	0	0	c
a	0	0	0	d
0	0	b	0	0
0	0	0	0	0

Key contributions of this work

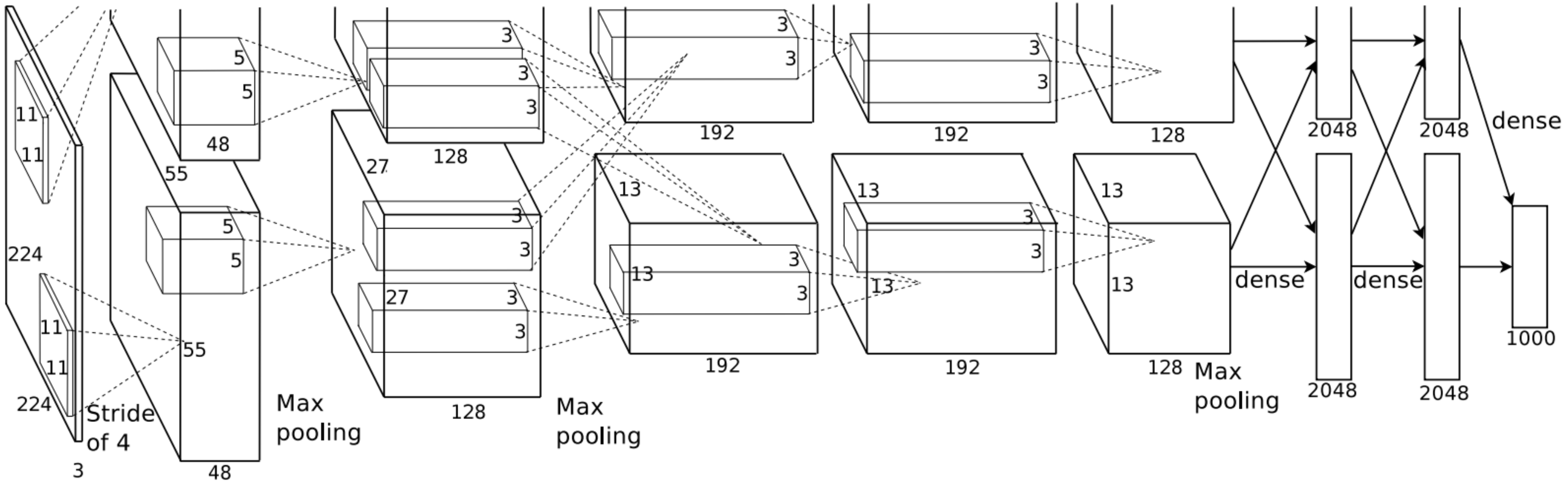
Application characterization study on sparsity when training convolutional neural networks

Architectural support for leveraging activation sparsity in virtualized DNNs

Q. How much sparsity do DNNs exhibit during training?

Case study) AlexNet

Characterizing the changes in layer density during training



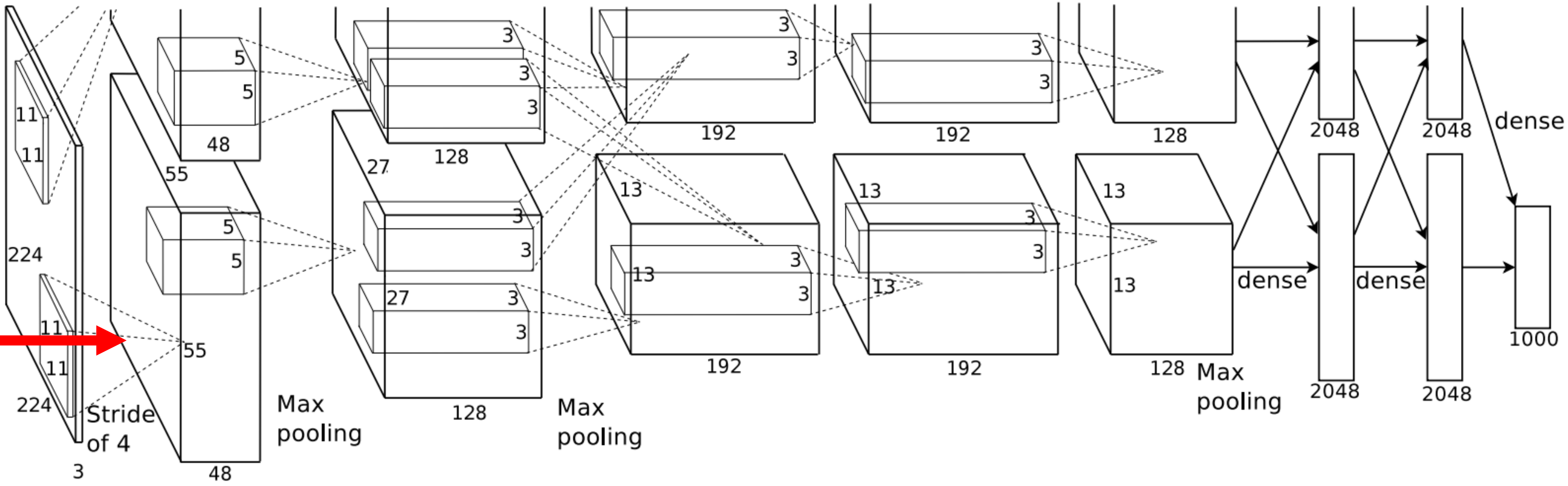
[AlexNet*]

* Krizhevsky et al., “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS-2012

Case study) AlexNet

Characterizing the changes in layer density during training

Test image



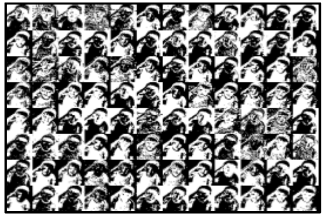
[AlexNet*]

* Krizhevsky et al., “ImageNet Classification with Deep Convolutional Neural Networks”, NIPS-2012

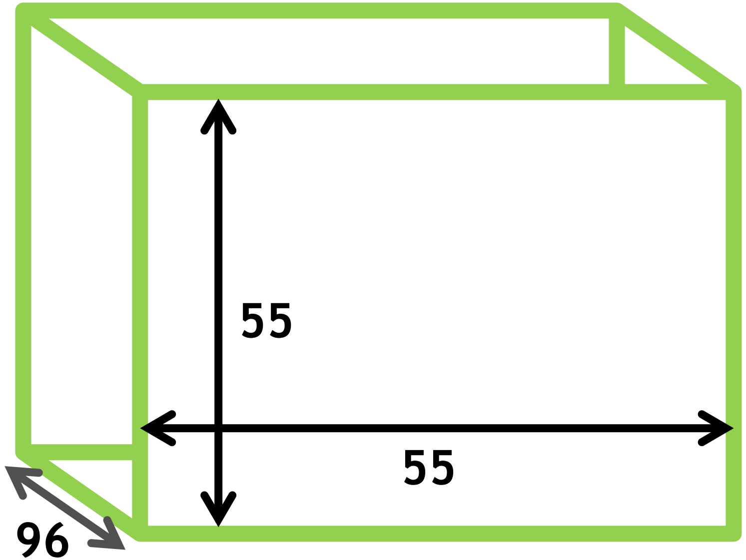
Case study) AlexNet

Characterizing the changes in layer density during training

conv0
(96, 55, 55)



Trained
(0%)



Feature maps

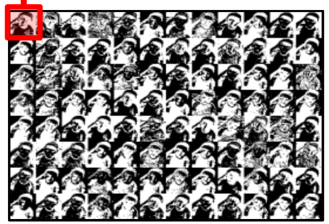
Test image

Case study) AlexNet

Characterizing the changes in layer density during training

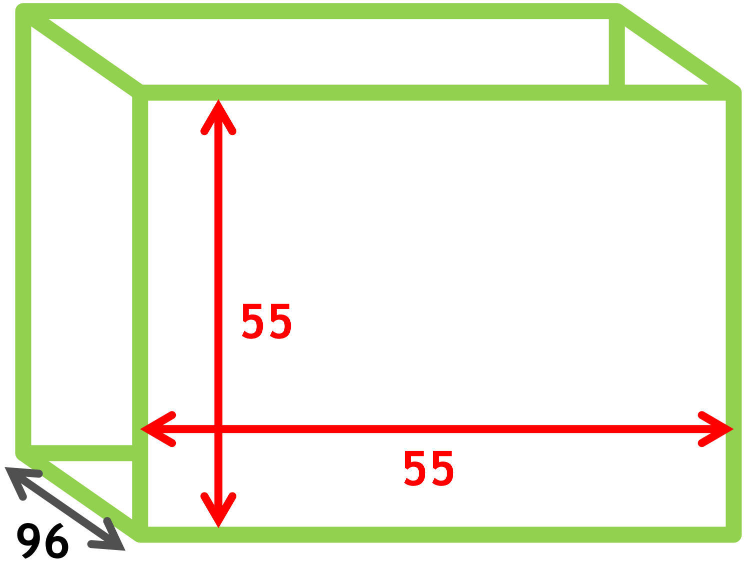
(55x55)
2D image

conv0
(96, 55, 55)



Trained
(0%)

Test image



Feature maps

Case study) AlexNet

Characterizing the changes in layer density during training

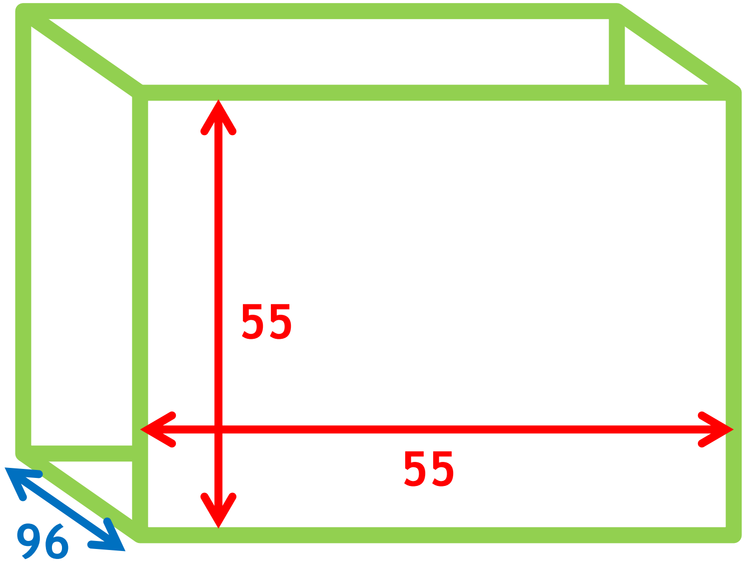
(55x55)
2D image

conv0
(96, 55, 55)

96 channels

Trained
(0%)

Test
image

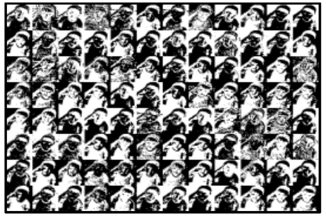


Feature maps

Case study) AlexNet

Characterizing the changes in layer density during training

conv0
(96, 55, 55)



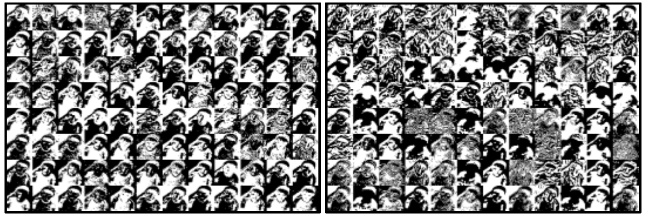
Trained
(0%)

Test
image

Case study) AlexNet

Characterizing the changes in layer density during training

conv0
(96, 55, 55)



Trained
(0%)

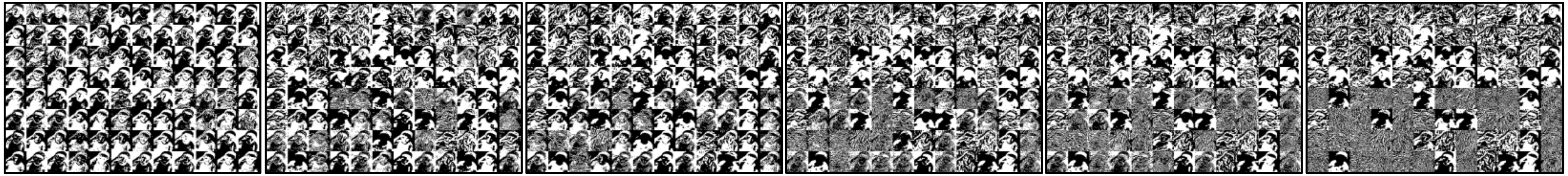
Trained
(20%)

Test image

Case study) AlexNet

Characterizing the changes in layer density during training

conv0
(96, 55, 55)



Trained (0%)

Trained (20%)

Trained (40%)

Trained (60%)

Trained (80%)

Trained (100%)

Test image

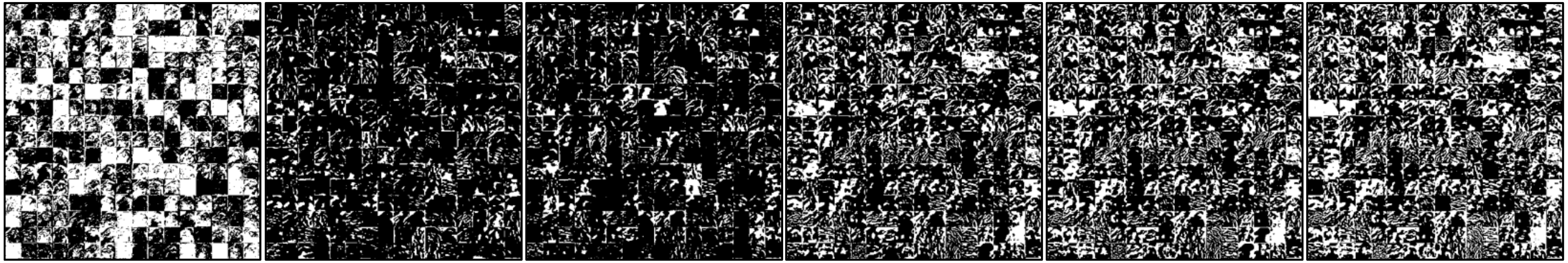
Average layer density: **49%**

(51% of activations are 0-valued)

Case study) AlexNet

Characterizing the changes in layer density during training

conv1
(256, 27, 27)



Trained (0%) | **Trained** (20%) | **Trained** (40%) | **Trained** (60%) | **Trained** (80%) | **Trained** (100%)

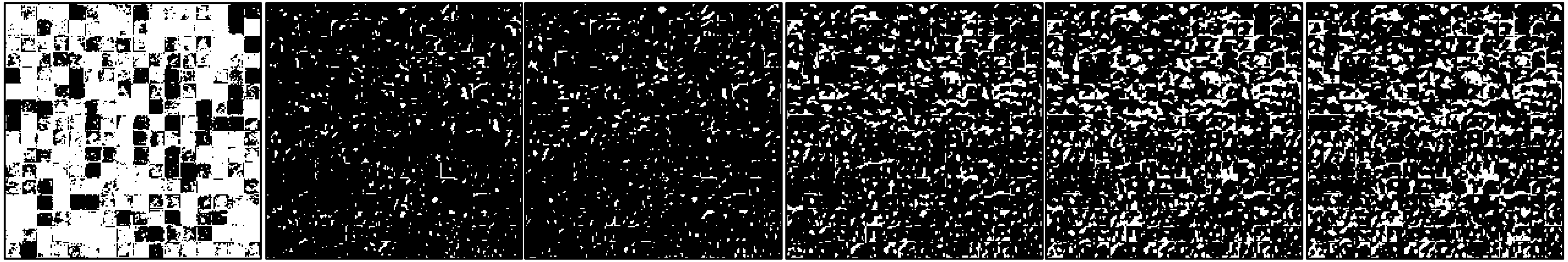
Average layer density: **36%**

(64% of activations are 0-valued)

Case study) AlexNet

Characterizing the changes in layer density during training

conv4
(256, 13, 13)



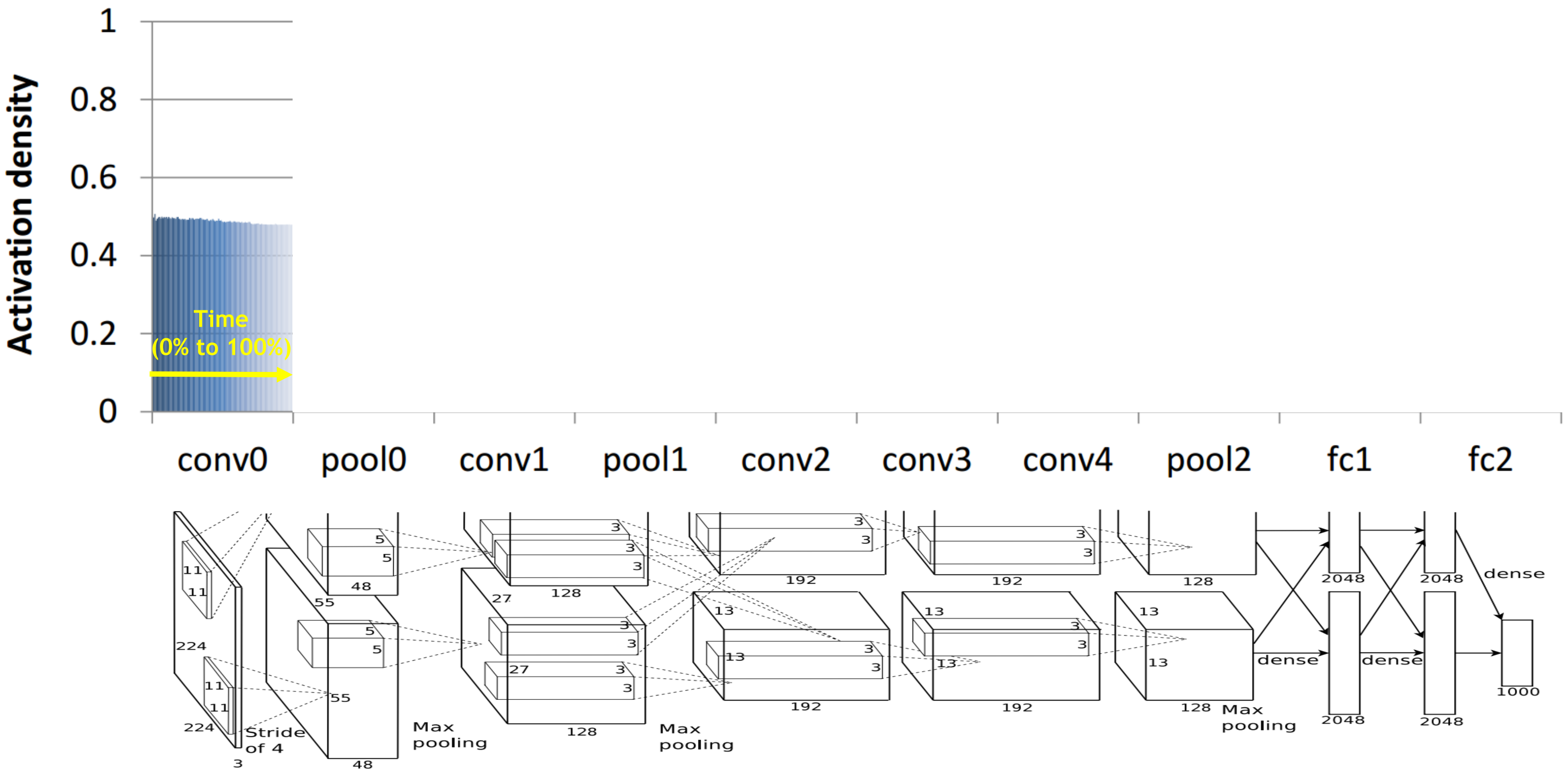
Trained (0%) | **Trained** (20%) | **Trained** (40%) | **Trained** (60%) | **Trained** (80%) | **Trained** (100%)

Average layer density: **22%**

(78% of activations are 0-valued)

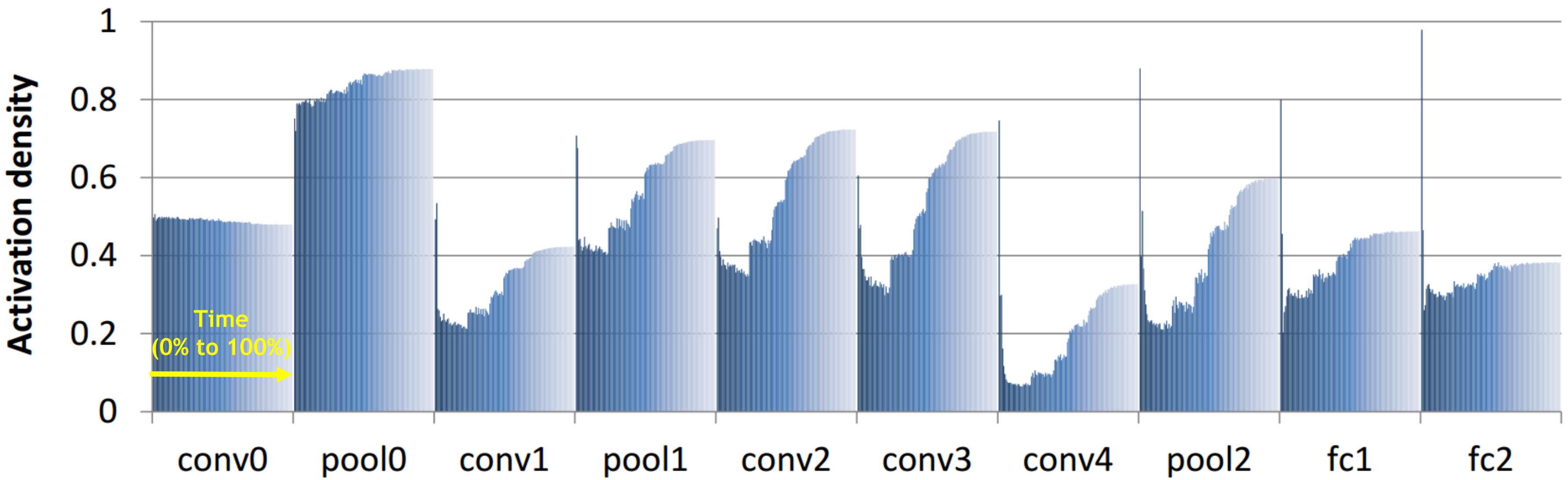
Case study) AlexNet

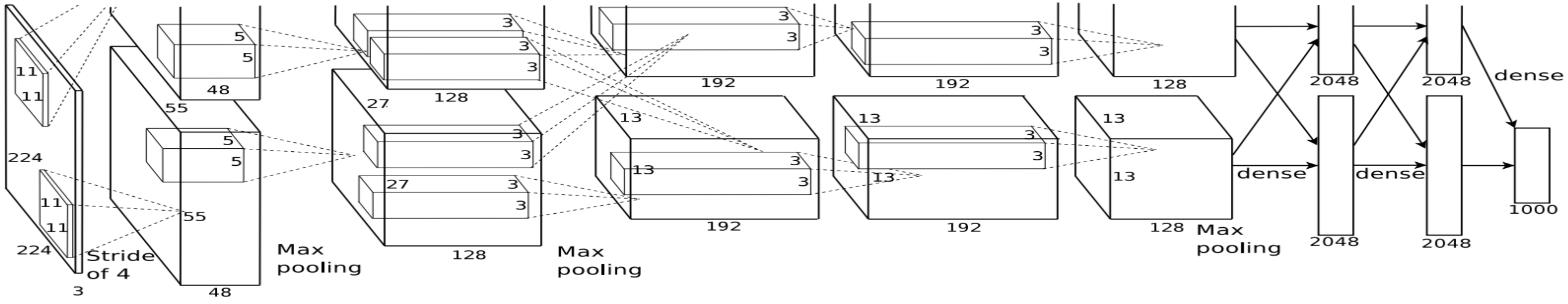
Putting everything together



Case study) AlexNet

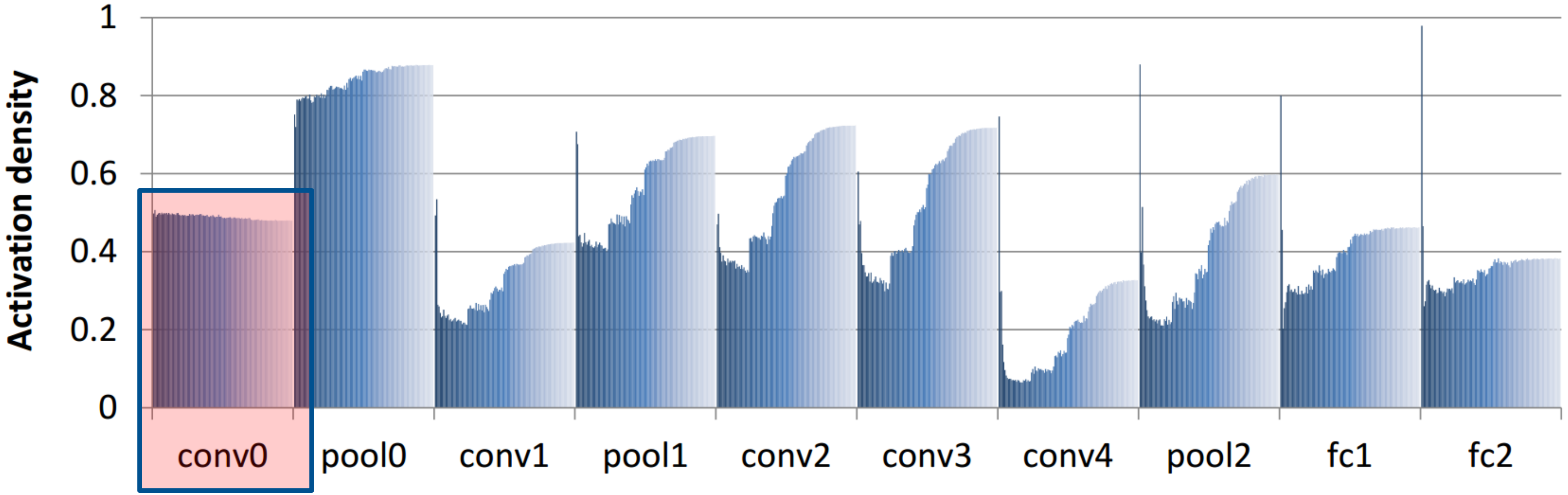
Putting everything together





Case study) AlexNet

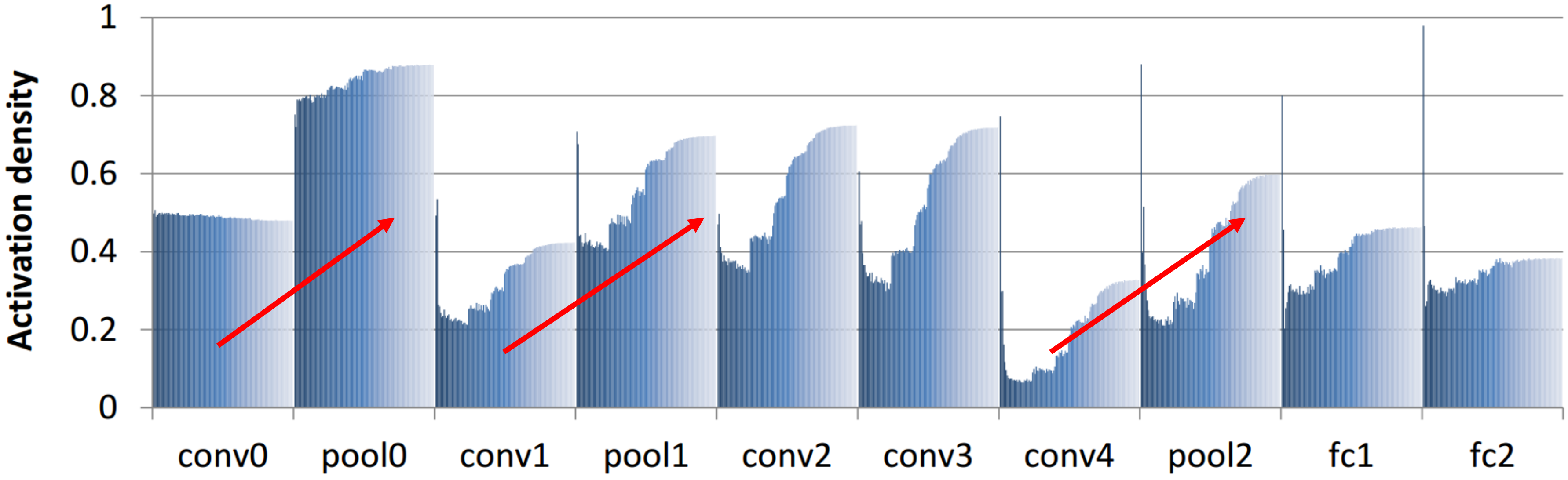
Putting everything together



Observation #1: First CONV layer consistently exhibits around 50% layer density across the entire training process.

Case study) AlexNet

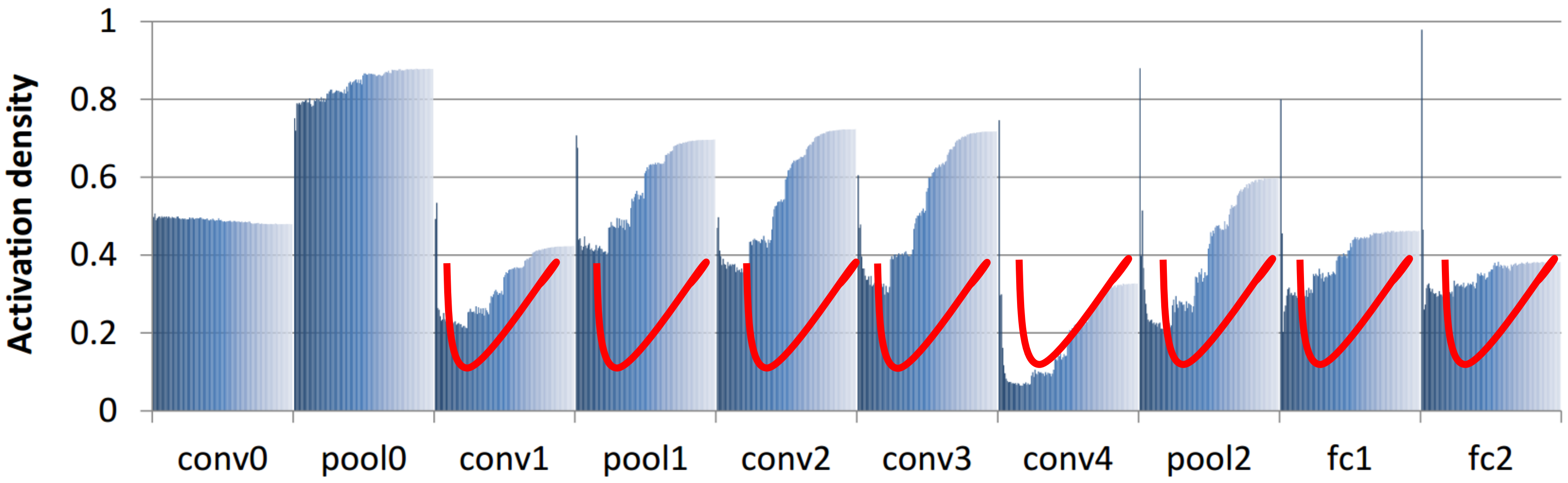
Putting everything together



Observation #2: Pooling layers always increase overall activation density.

Case study) AlexNet

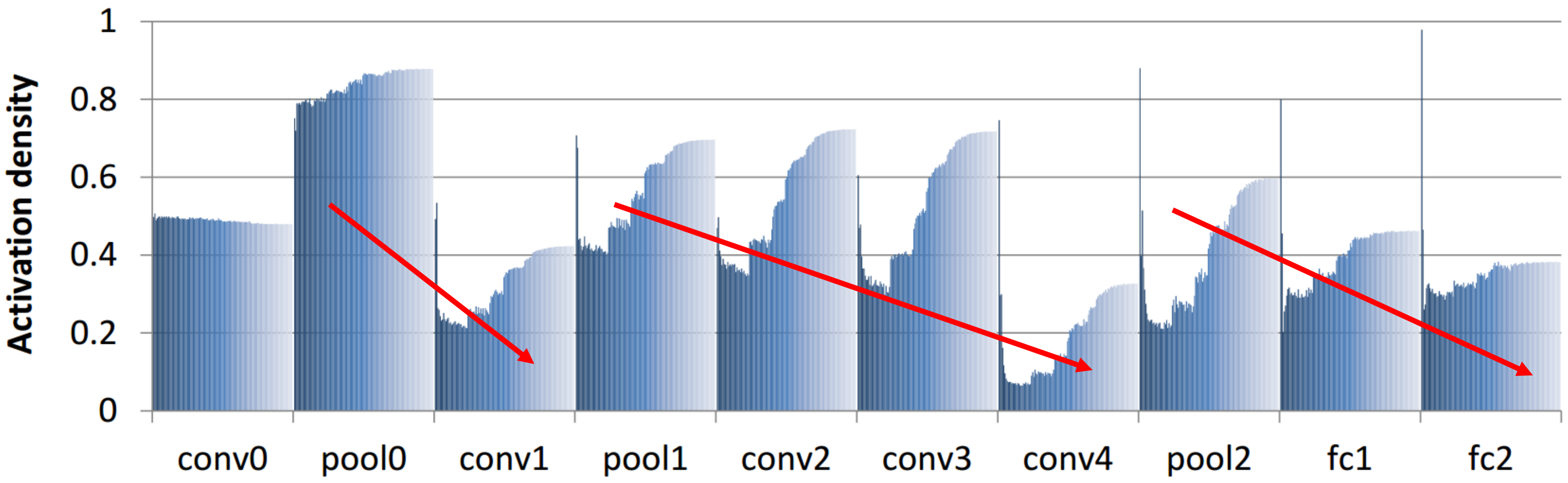
Putting everything together



Observation #3: Within each layer, activation density rapidly decreases during the initial training periods; once training period reaches the fine-tuning stage, density gradually crawls back up again.

Case study) AlexNet

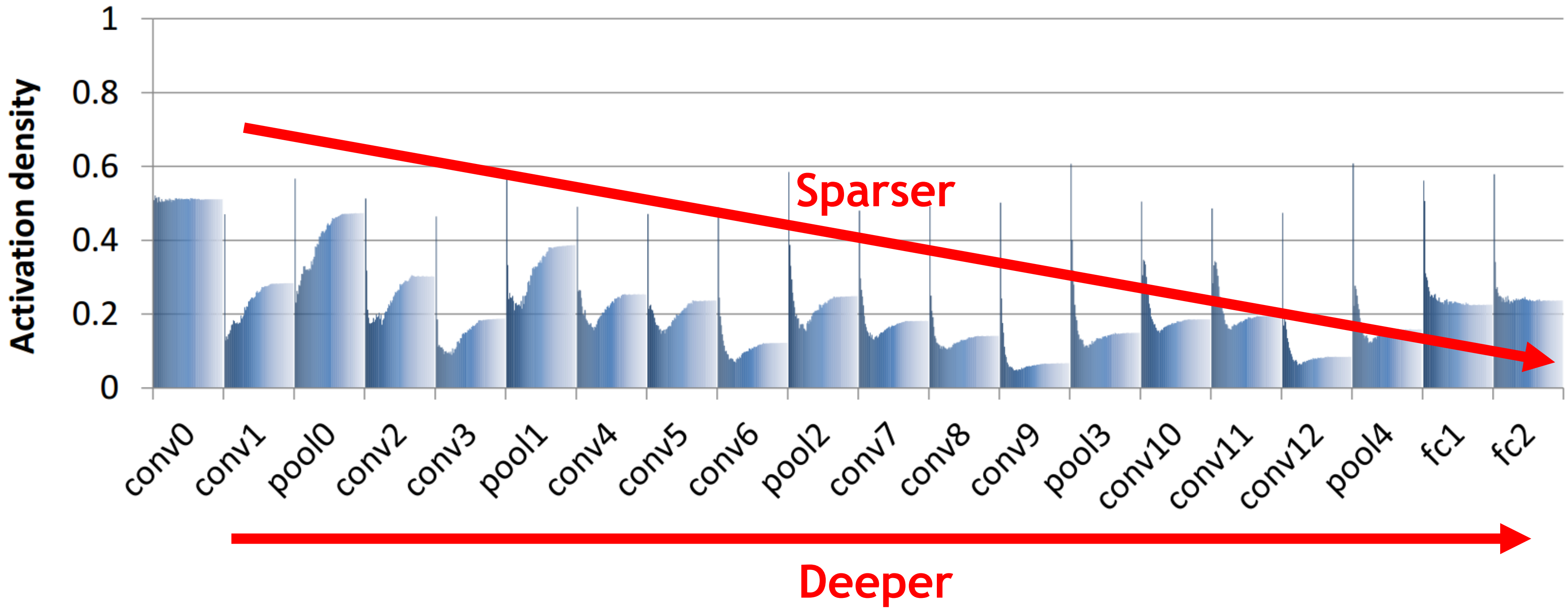
Putting everything together



Observation #4: Later layers are generally more sparser than earlier layers

Case study) VGG-16

Putting everything together

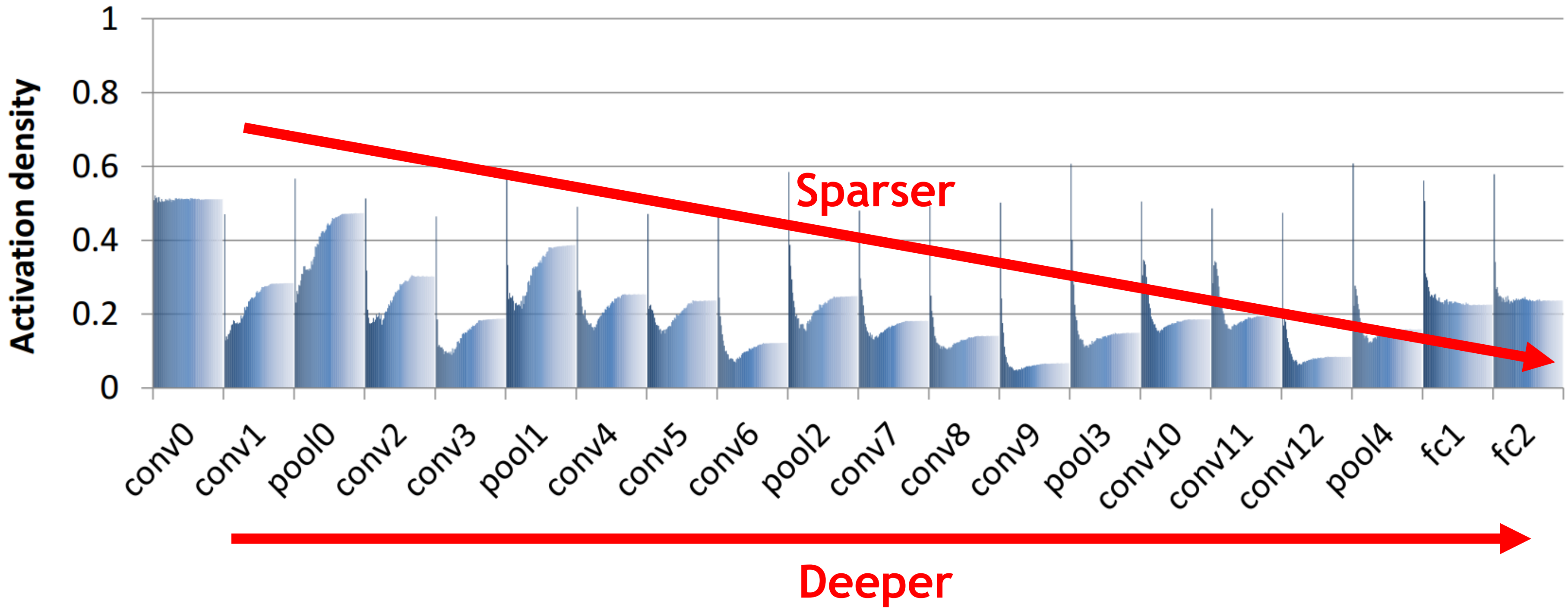


What causes such behavior in DNNs?

Discussed much more in our paper ☺

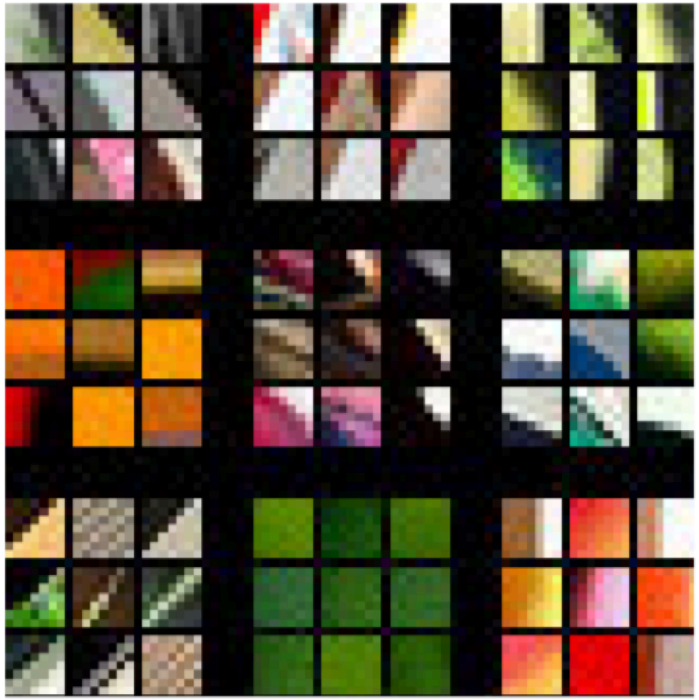
What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network



What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network

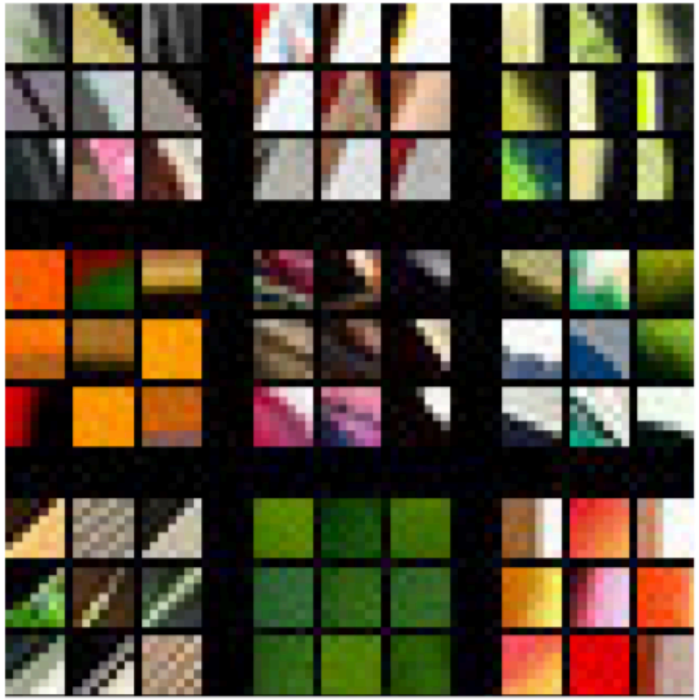

Input images


Activations

* Zeiler et al., "Visualizing and Understanding Convolutional Networks", arXiv.org, 2013

What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network



Input images

Activations

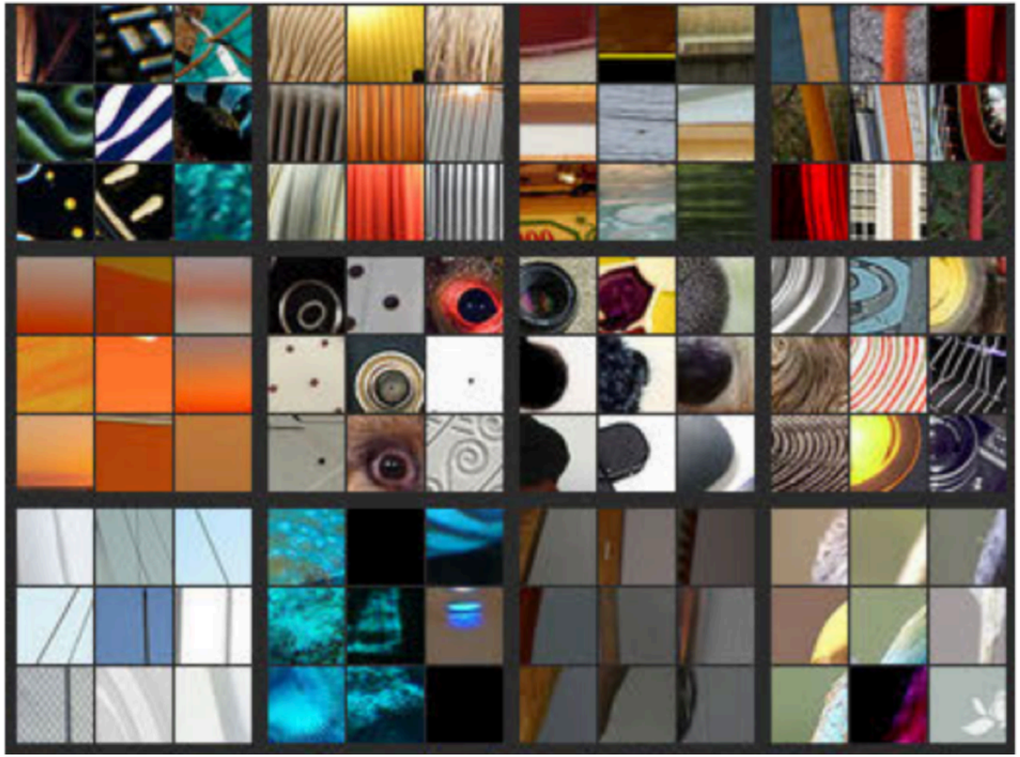
First few layers: filters are trained to respond to **"class-invariant"** features

- Corners
- Edges
- Colors

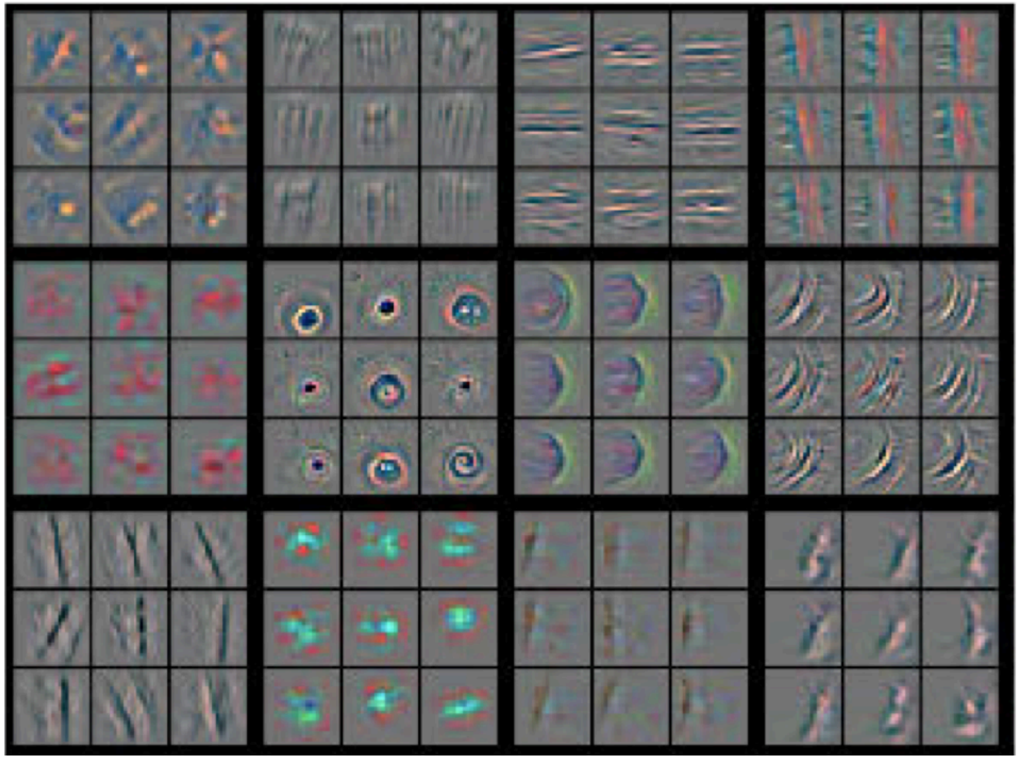
* Zeiler et al., "Visualizing and Understanding Convolutional Networks", arXiv.org, 2013

What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network



Input images

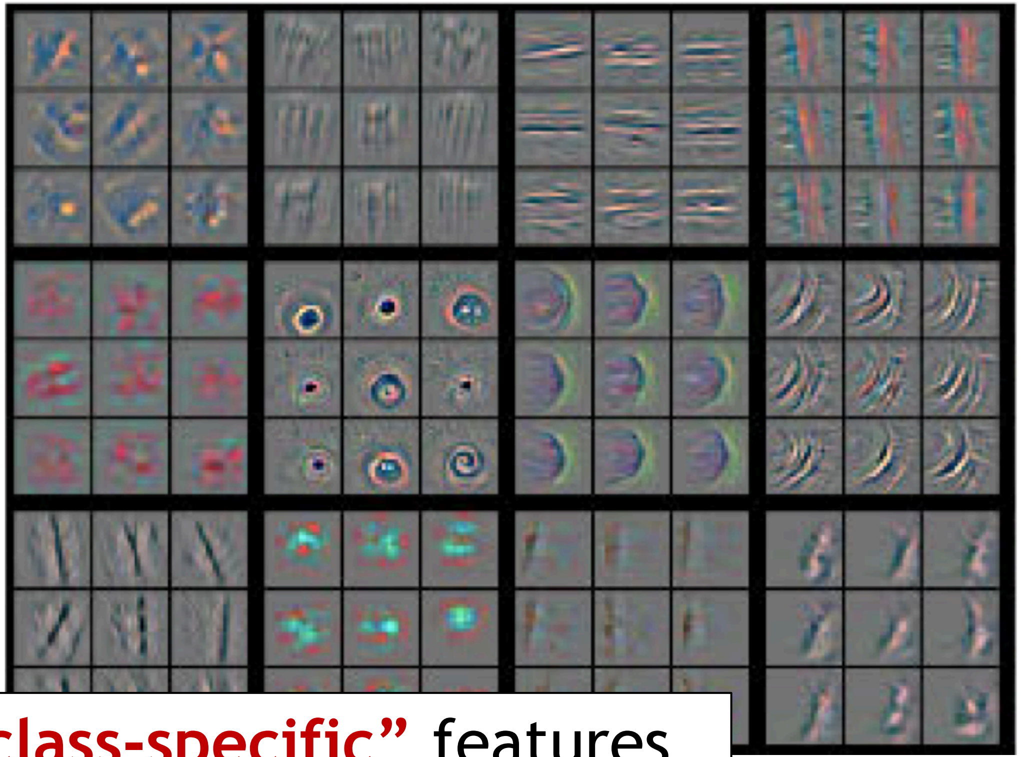


Activations

* Zeiler et al., "Visualizing and Understanding Convolutional Networks", arXiv.org, 2013

What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network

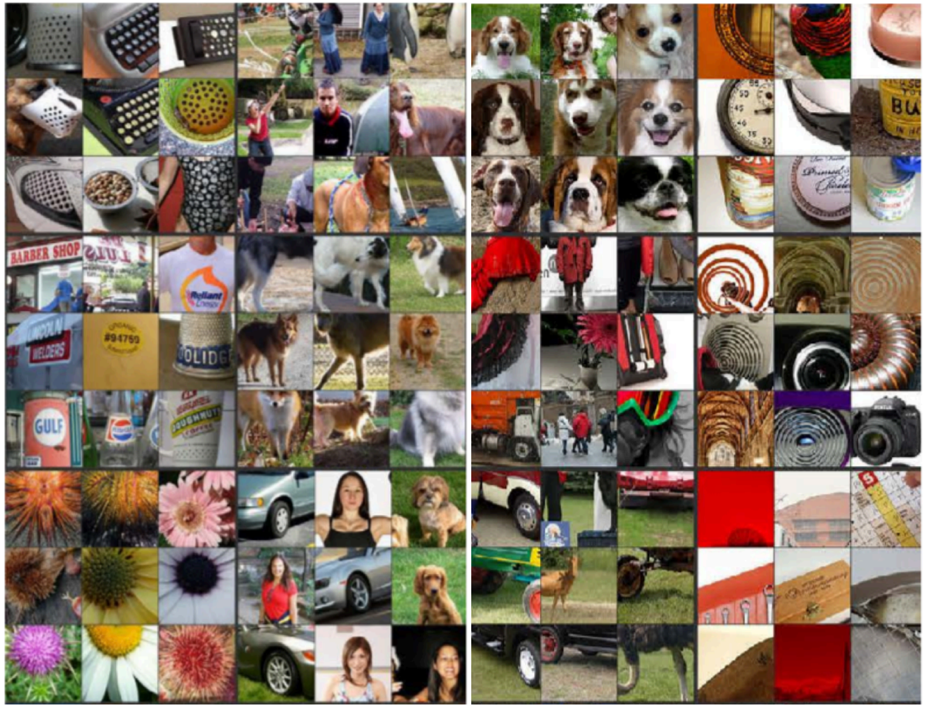


Deeper layers: more “**class-specific**” features (e.g., Textures …)

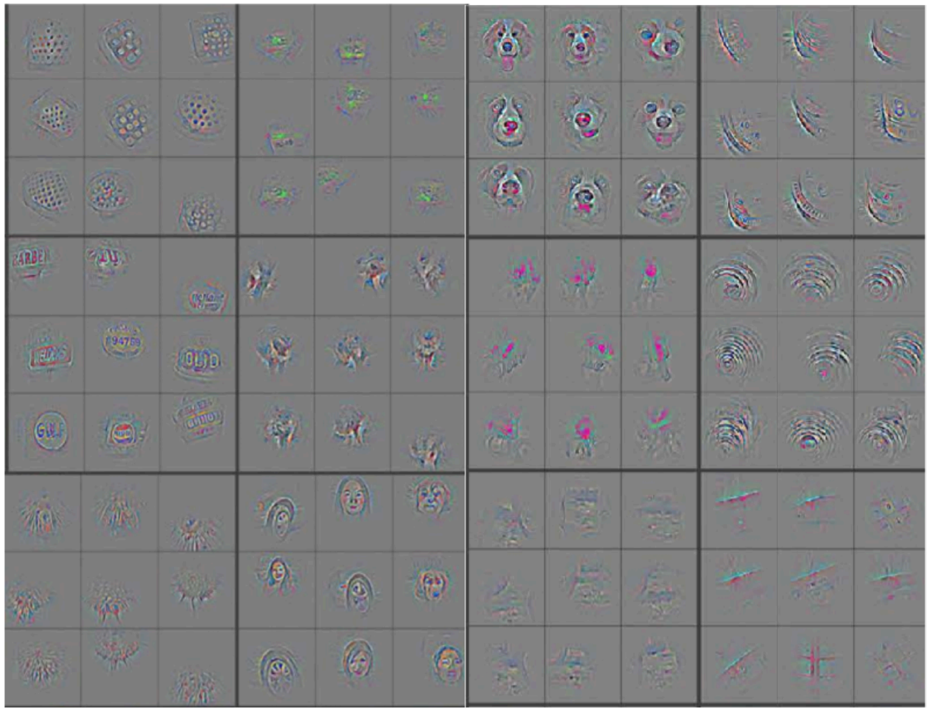
* Zeiler et al., “Visualizing and Understanding Convolutional Networks”, arXiv.org, 2013

What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network



Input images

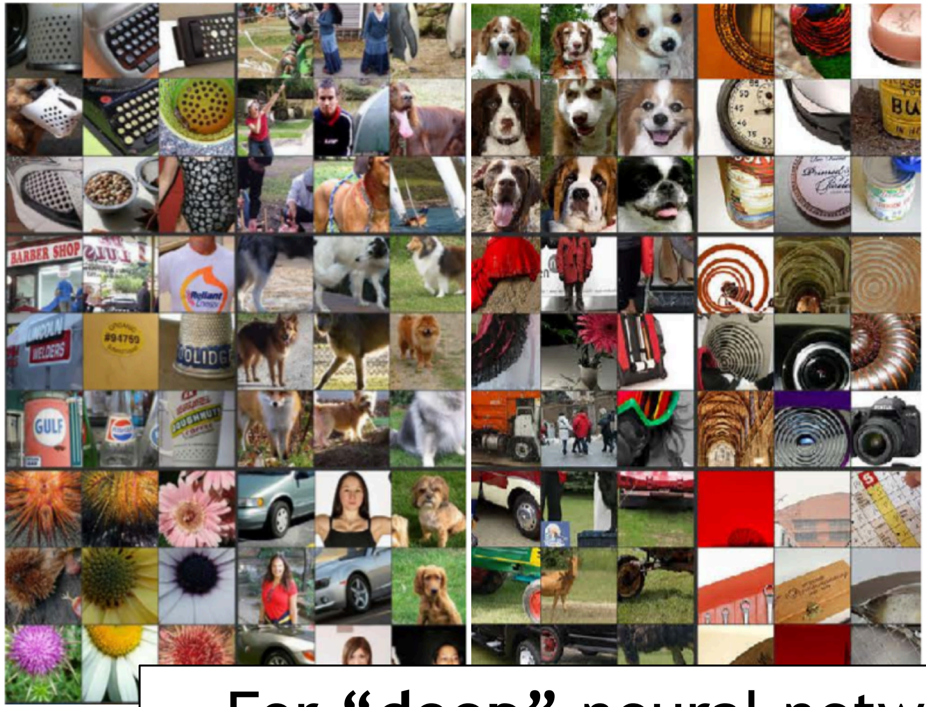


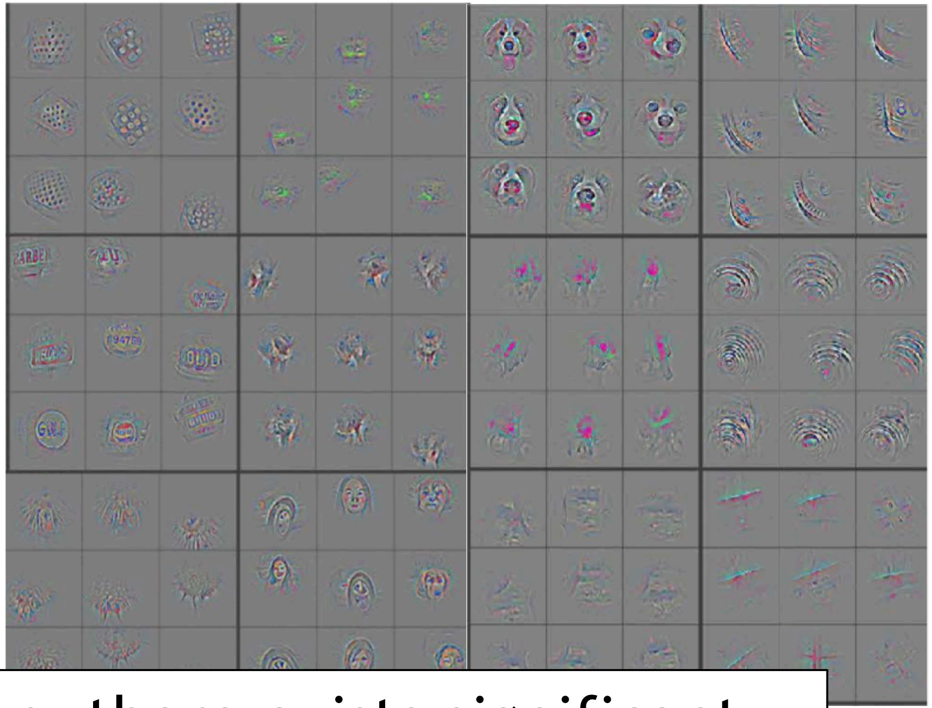
Activations

* Zeiler et al., “Visualizing and Understanding Convolutional Networks”, arXiv.org, 2013

What causes such behavior in DNNs?

Observation#4: Sparsity increases as you go deep inside the network





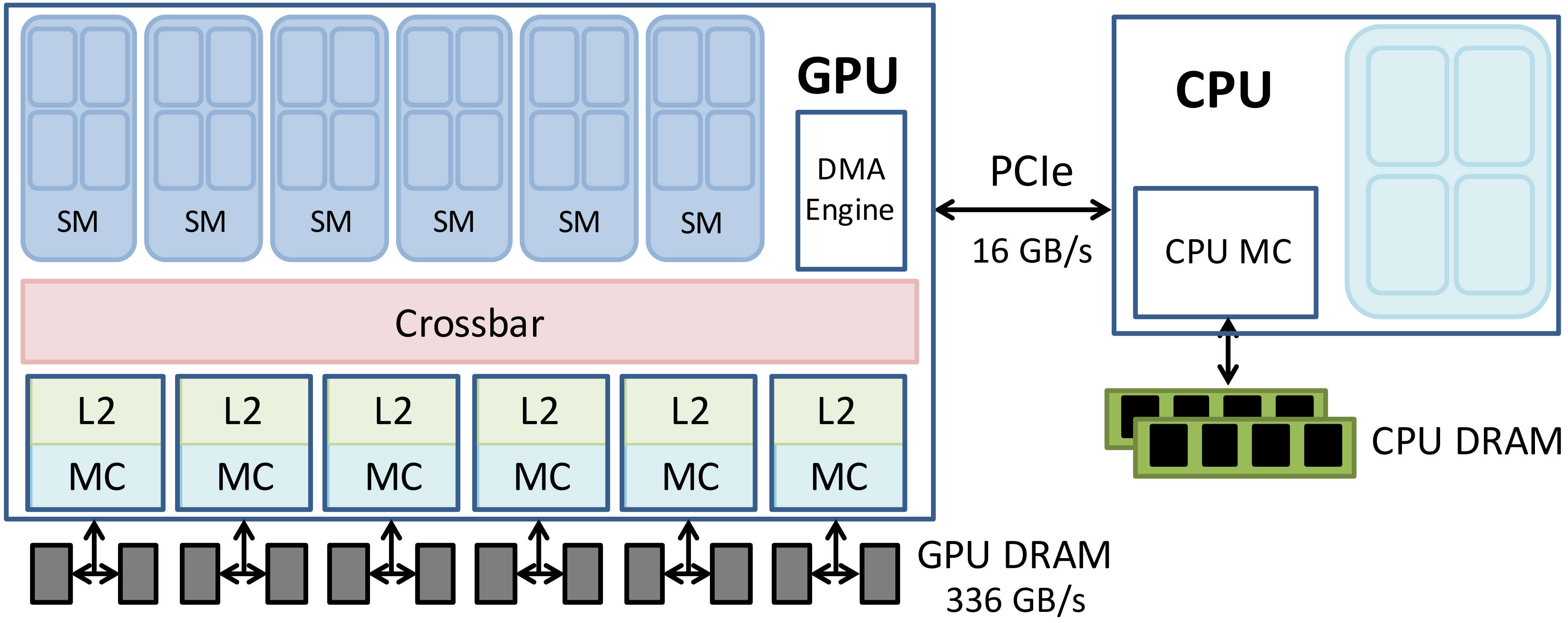
For "**deep**" neural networks, there exists significant sparsity in activations (**40% ~ 90% layer-wise sparsity**)

* Zeiler et al., "Visualizing and Understanding Convolutional Networks", arXiv.org, 2013

Compressing DMA Engine (cDMA)

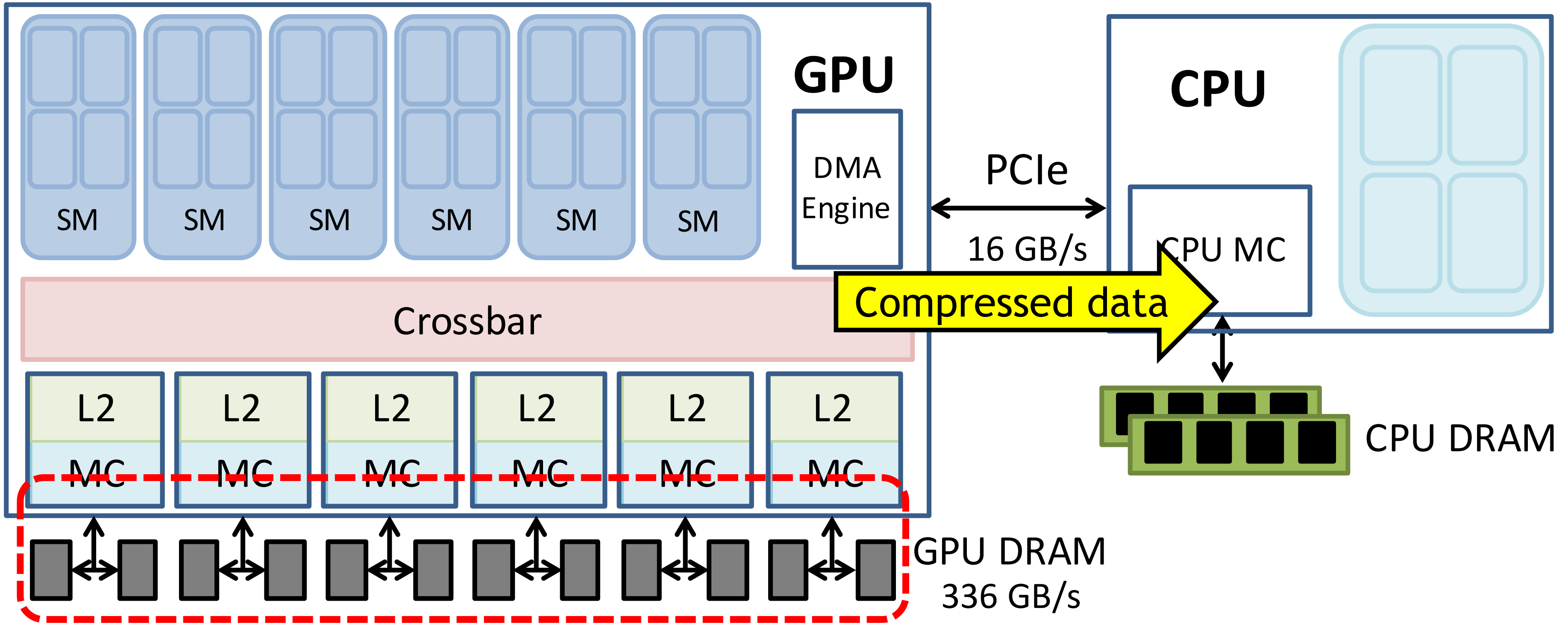
Baseline CPU-GPU system interconnect

Max. 16 GB/sec communication channel between CPU-GPU



Compressing DMA architecture

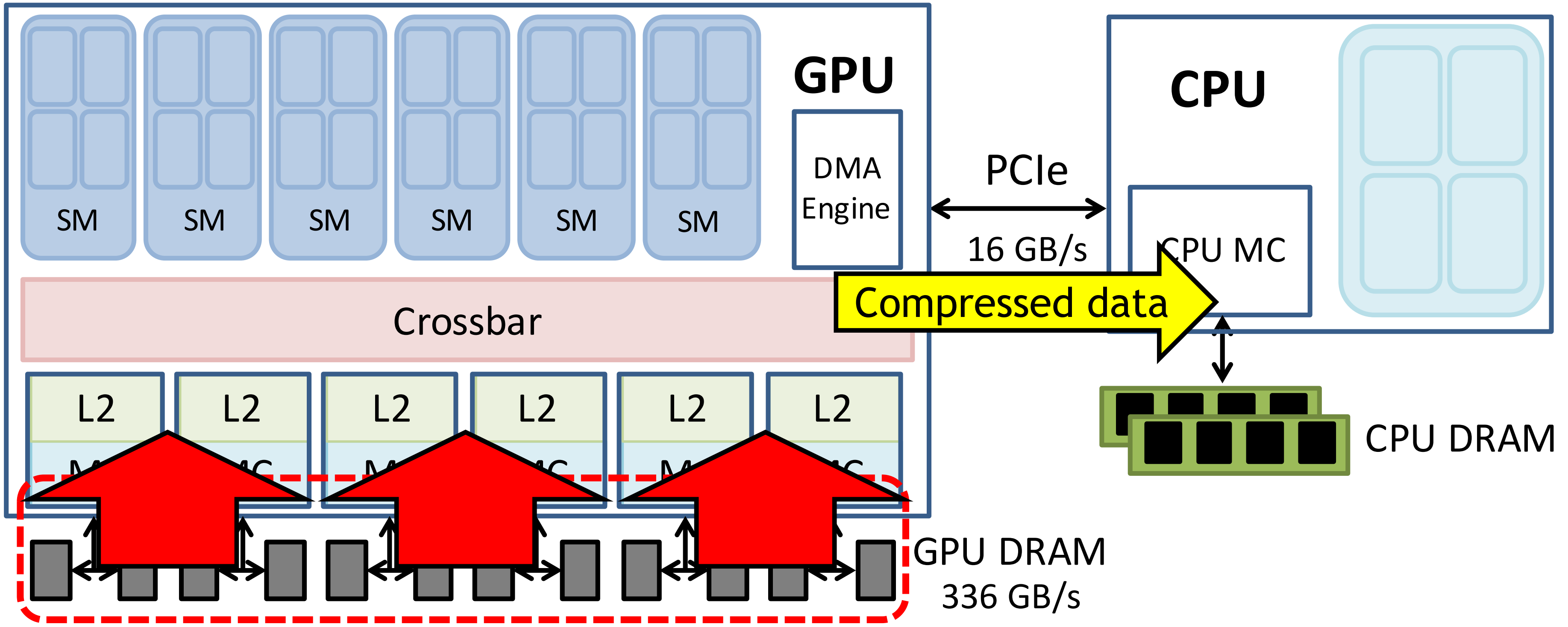
Goals: Saturate PCIe channel with compressed activation maps



Q. How should the memory subsystem interact with the DMA engine?

Compressing DMA architecture

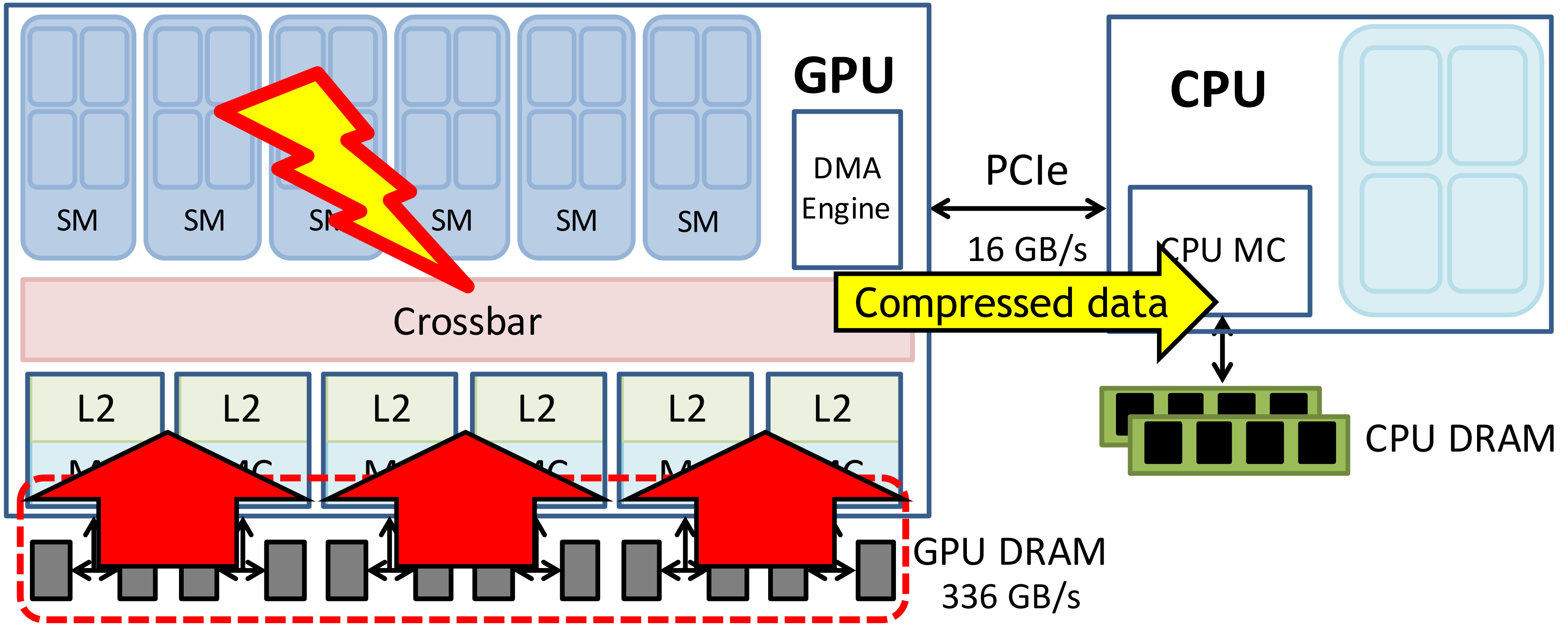
DRAM read-BW should be high enough to generate compressed data



: DRAM read throughput >= (compression rate x PCIe bandwidth)

Compressing DMA architecture

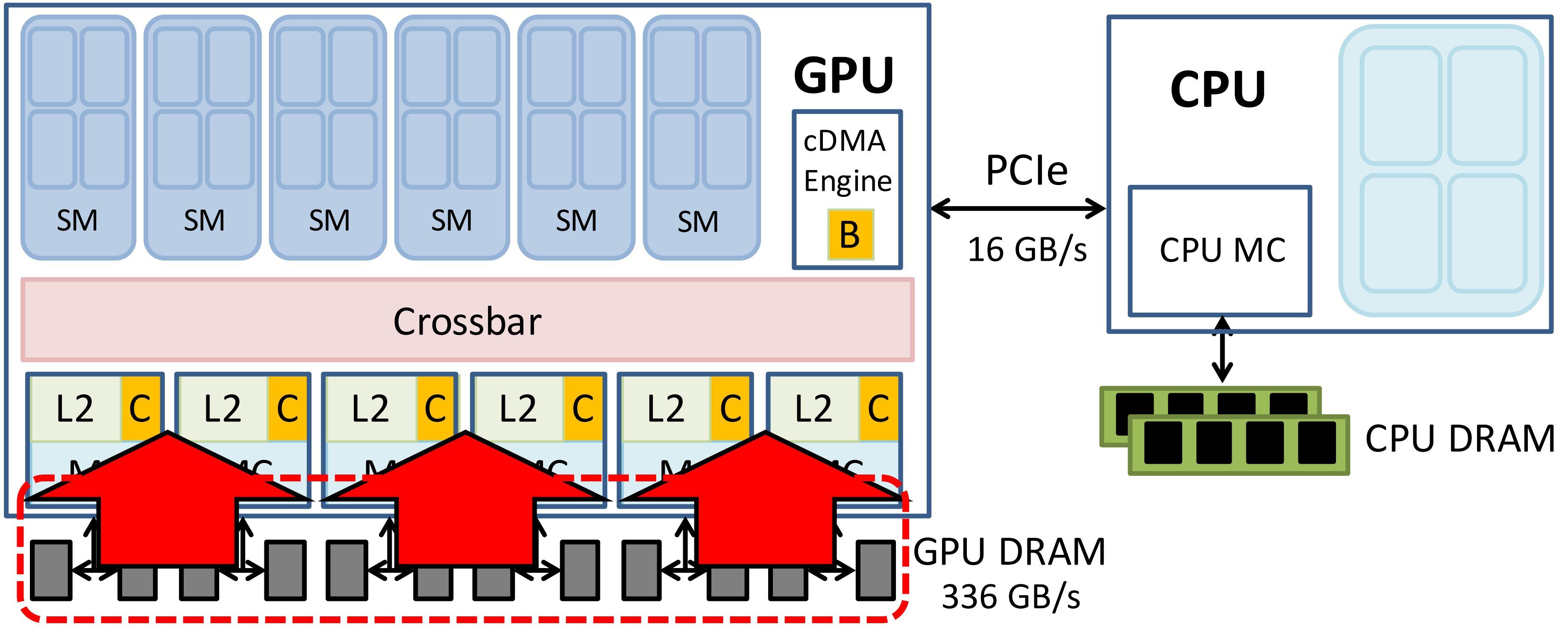
Challenges: GPU crossbar bandwidth should be amplified proportionally



: DRAM read throughput >= (compression rate x PCIe bandwidth)

Compressing DMA architecture

Solution: Compress data "before" routing it through the crossbar

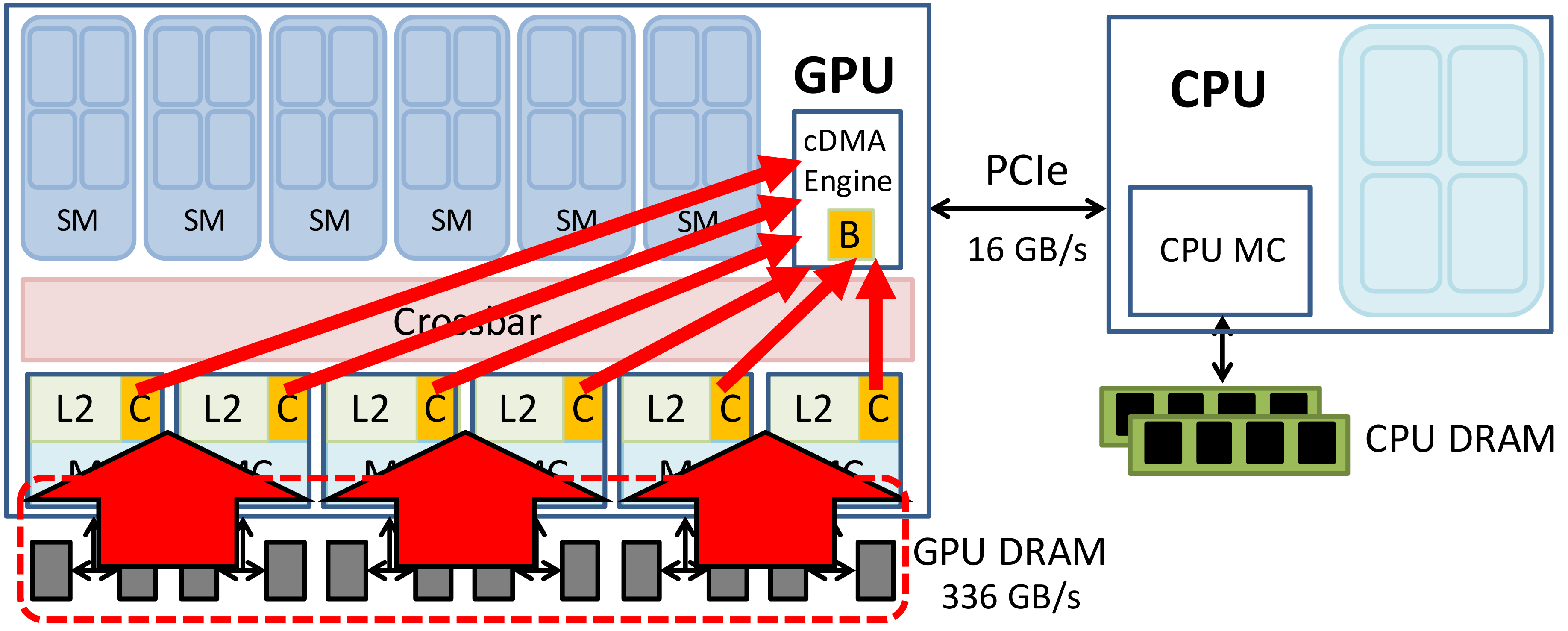


C : Compression unit

B : Buffer to aggregate compressed data from all MCs

Compressing DMA architecture

Solution: Compress data “before” routing it through the crossbar



C : Compression unit

B : Buffer to aggregate compressed data from all MCs

Compression algorithms

Compression algorithms

1. Run-length encoding

+ Simple to implement, well-suited for high-throughput compression

-- Compression rate is good only when zero-values are clustered

2. Zlib compression

+ Exhibits good compression rate for a variety of data patterns

-- Designing high-throughput compression hardware is challenging

e.g., Dedicated ASIC/FPGA solutions provide roughly 2.5 GB/sec data

Proposed compression algorithm

Frequent-value compression (encoding sparseness)

< Uncompressed >

Data	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p

< Compressed >

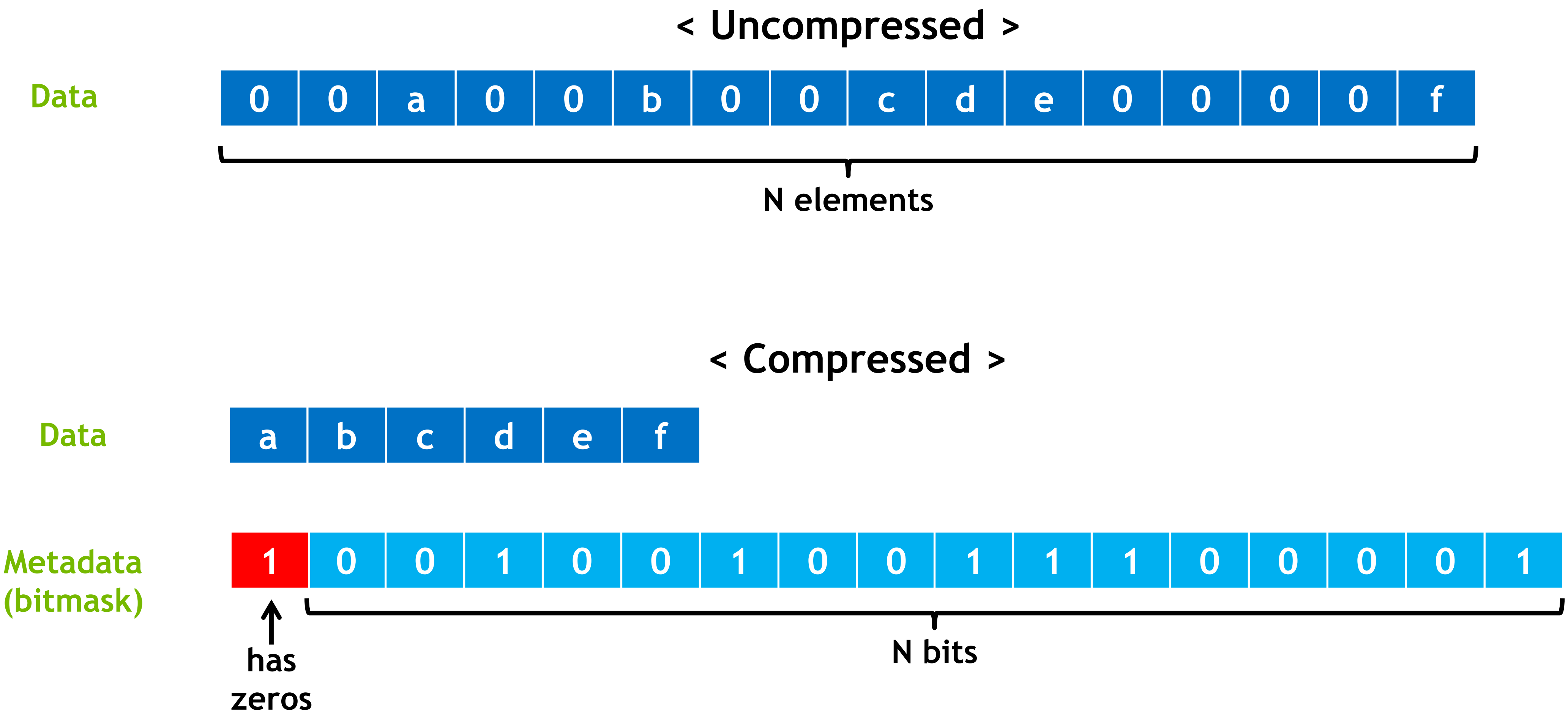
Data	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p

Metadata	0

↑
has zeros

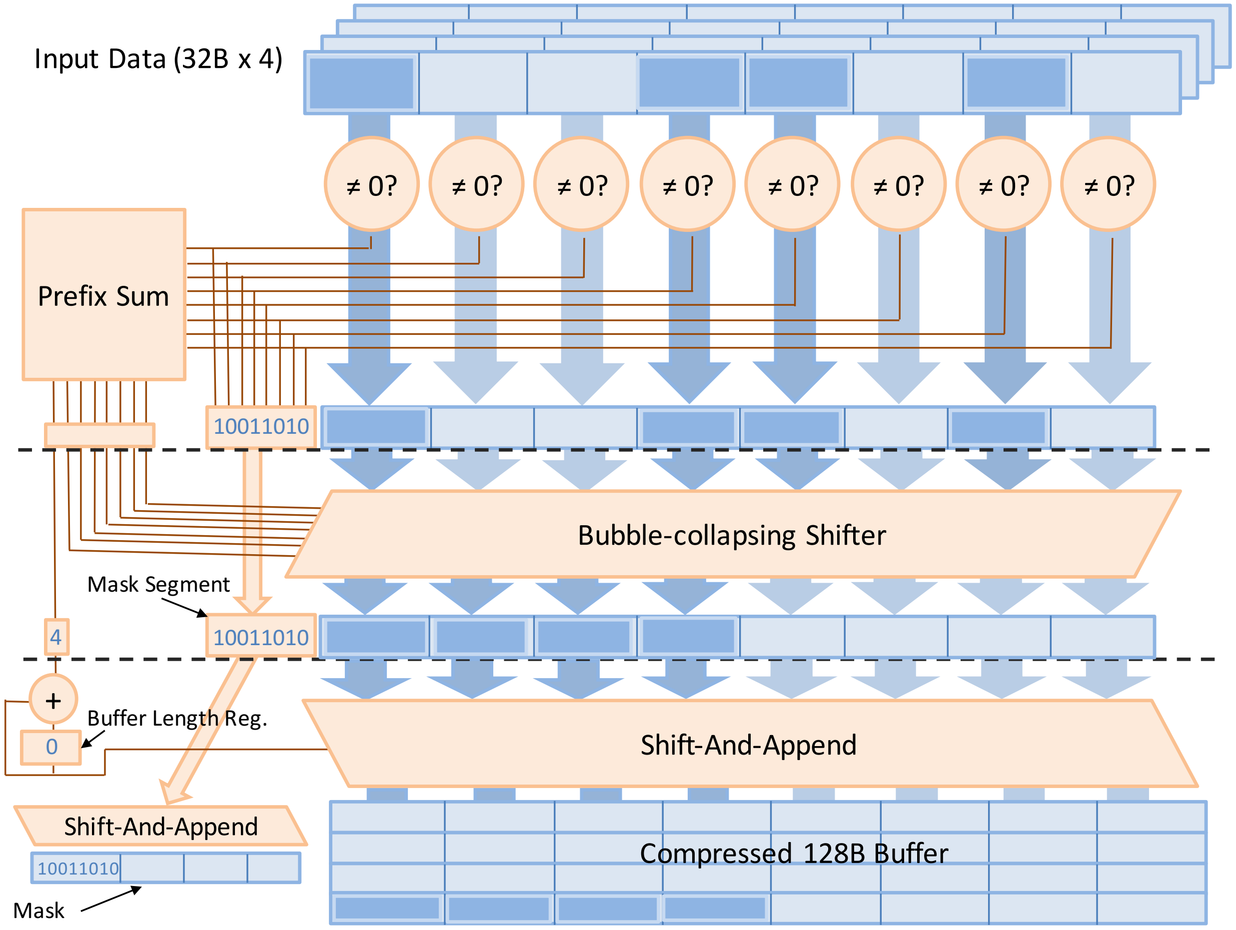
Proposed compression algorithm

Frequent-value compression (encoding sparseness)



Compression microarchitecture

Frequent-value compression (encoding sparseness)



<Area overhead>

- FreePDK + CACTI
- **1.5 mm^2** in 28 nm process
- (Note) GV100 size: 800 mm^2

Results

Evaluation

Methodology

Application characterization & datasets

- Model: trained from scratch using Caffe
- Activations: collected at training time, which are fed into the compression module

Evaluation

Methodology

Application characterization & datasets

- Model: trained from scratch using Caffe
- Activations: collected at training time, which are fed into the compression module

Performance evaluation (hybrid approach)

- Real GPU:
- Analytical model:

Evaluation

Methodology

Application characterization & datasets

Model: trained from scratch using Caffe

Activations: collected at training time, which are fed into the compression module

Performance evaluation (hybrid approach)

Real GPU:

measured using vDNN* with CPU-migrated data properly compressed

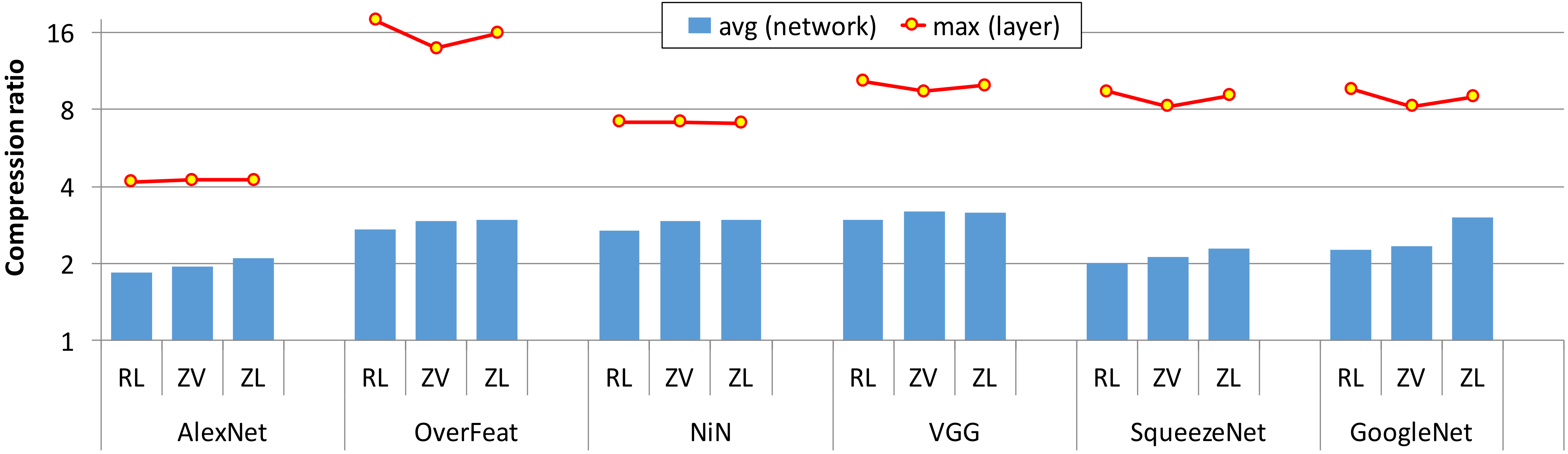
Analytical model:

penalize performance when cDMA's DRAM bandwidth pressure is high

* Rhu et al.,"vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design", MICRO-2016

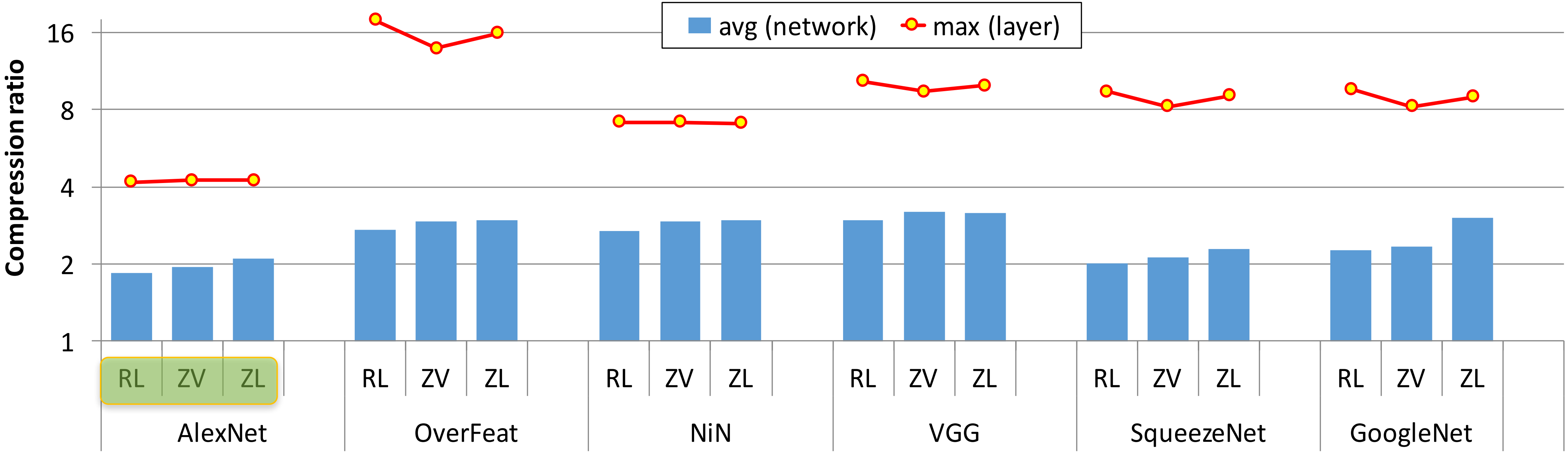
Avg/Max compression rate

Higher is better



Avg/Max compression rate

Higher is better

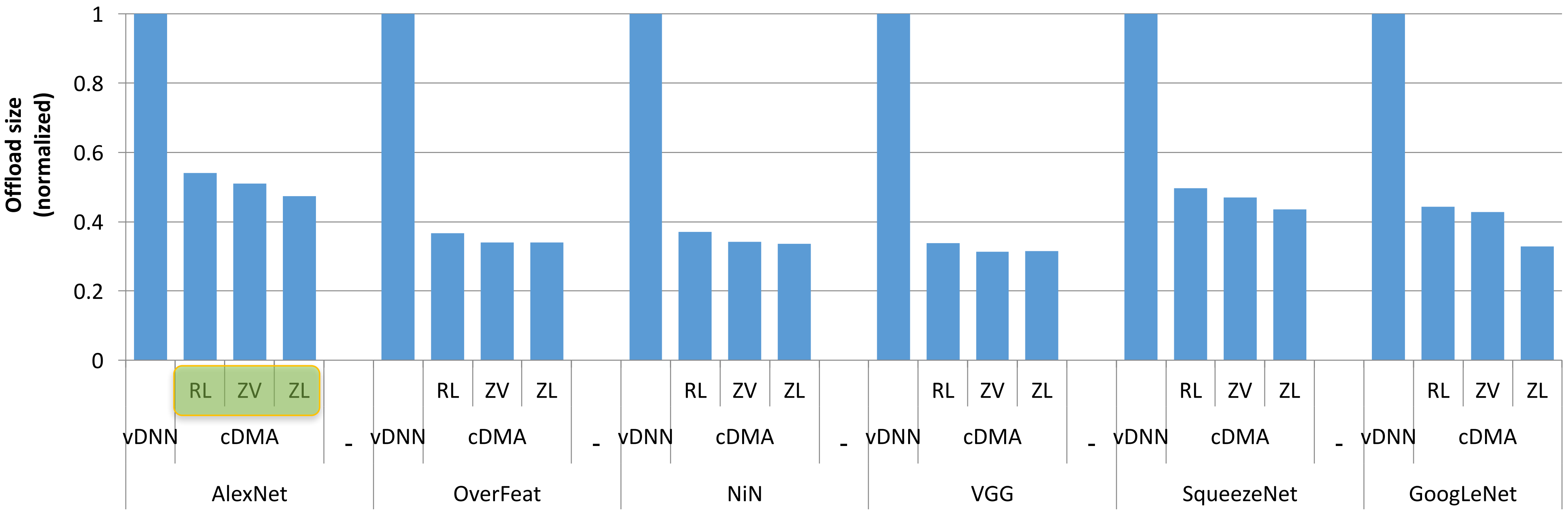


: different compression algorithm

➔ RL (run-length encoding), **ZV (zero-value compression)**, and ZL (Zlib compression)

CPU-GPU data traffic size

Lower is better

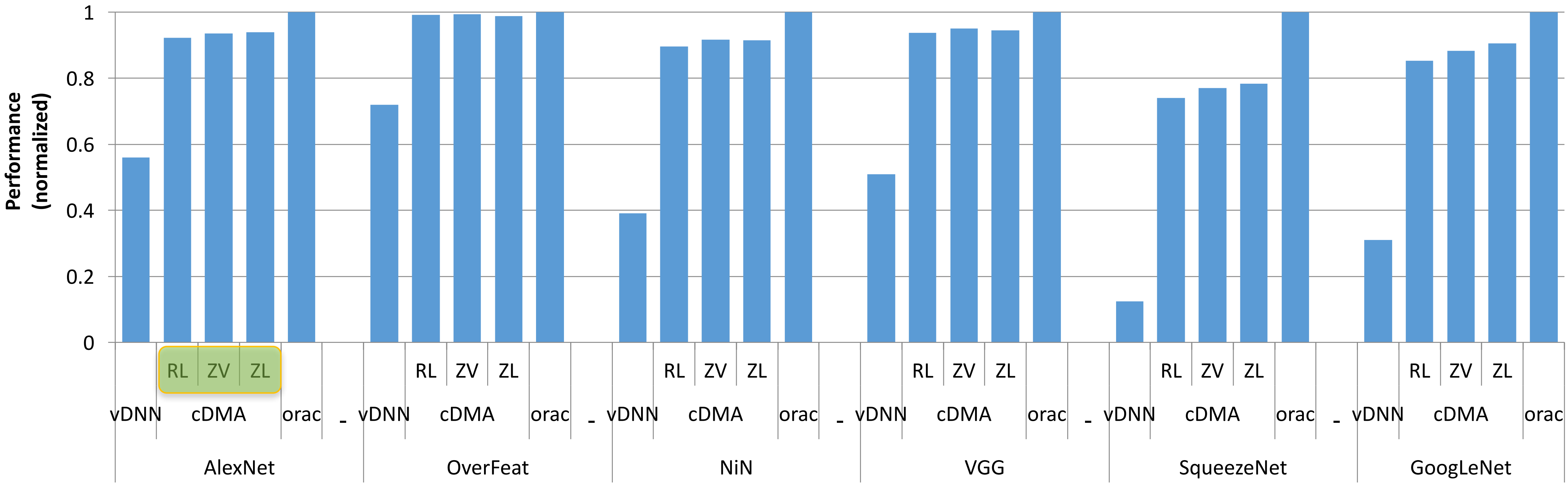


: different compression algorithm

➔ RL (run-length encoding), **ZV (zero-value compression)**, and ZL (Zlib compression)

Performance

Higher is better



: different compression algorithm

➔ RL (run-length encoding), **ZV (zero-value compression)**, and ZL (Zlib compression)

Conclusions

Compressing DMA engine:
Architectural support for sparse CNN training

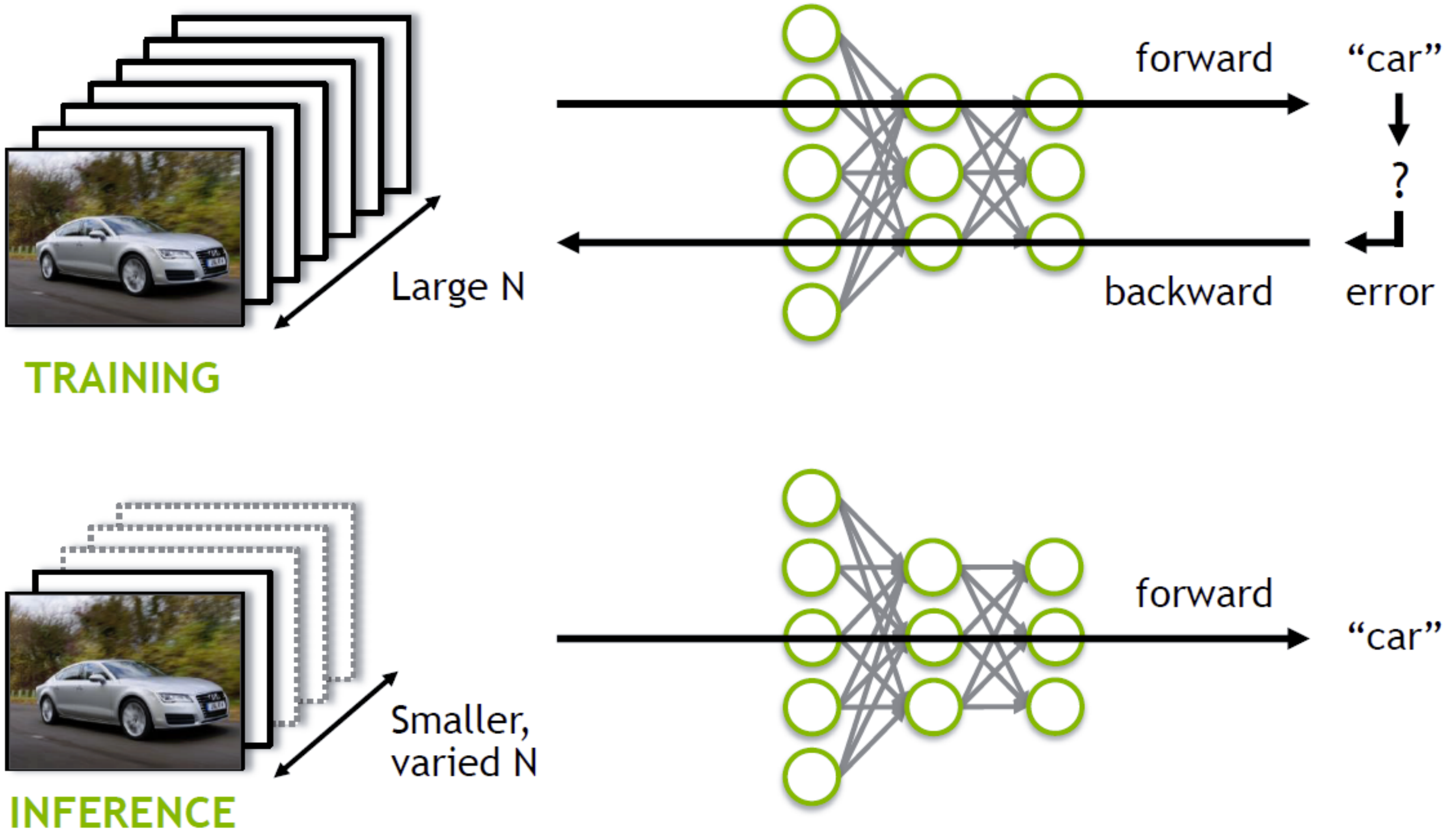
Avg 2.6x (max 13.8x) compression rate

Avg 53% (max 79%) speedup on Pascal Titan Xp

Backup

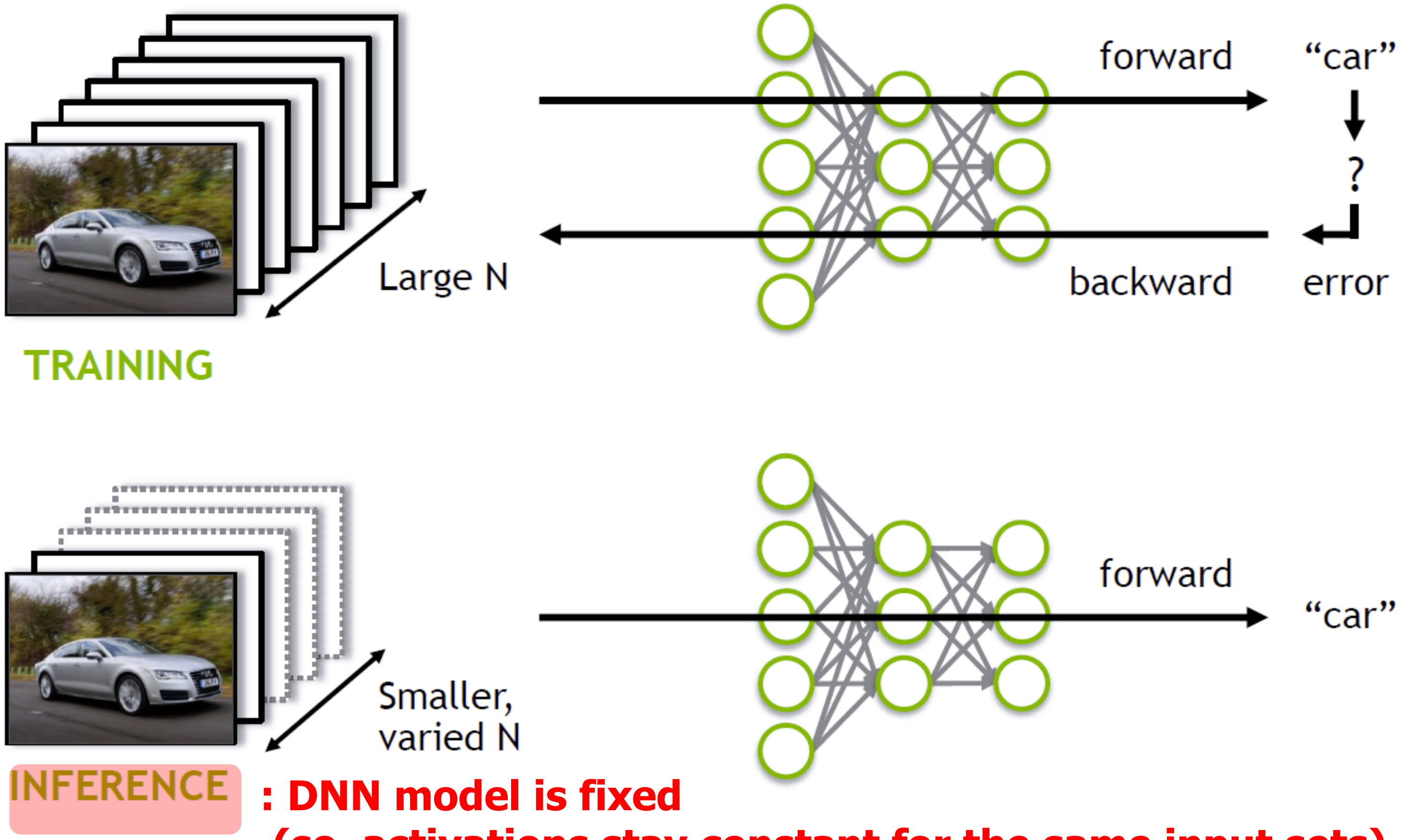
Training vs. inference

Deep learning for image classification



Training vs. inference

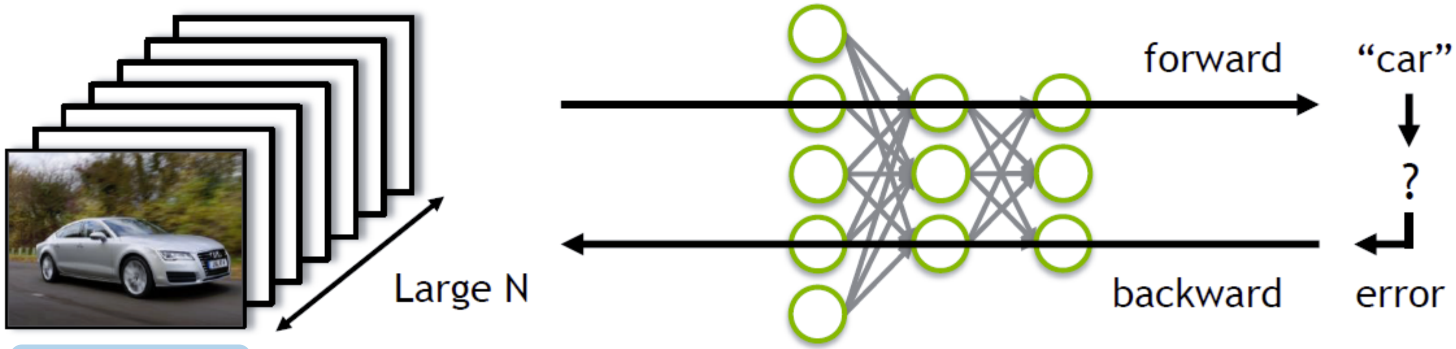
Deep learning for image classification



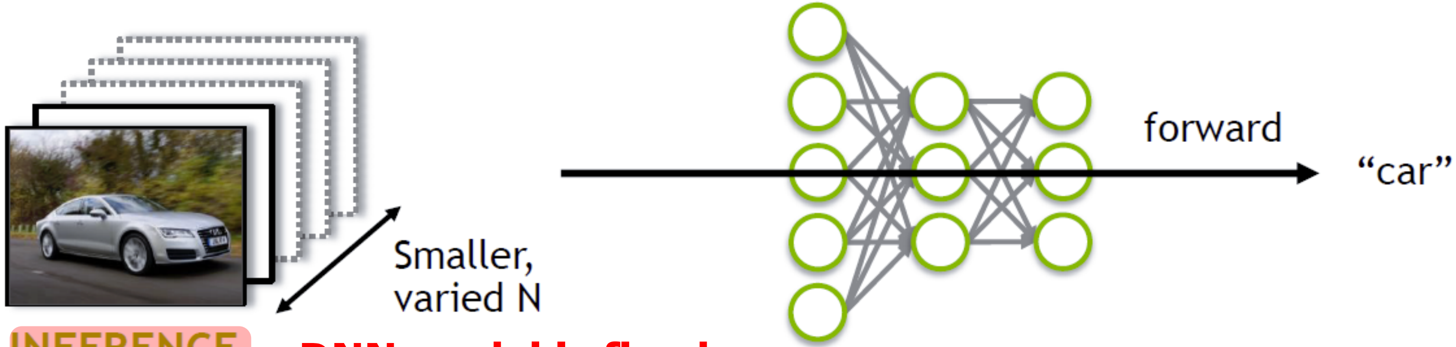
: DNN model is fixed
(so, activations stay constant for the same input sets)

Training vs. inference

Deep learning for image classification



TRAINING : **DNN model gets constantly updated during the course of training (so, activation map values also changes accordingly …)**



INFERENCE : **DNN model is fixed (so, activations stay constant for the same input sets)**

Case study) AlexNet

Characterizing the changes in layer density during training

fc1
(4096, 1, 1)

Trained (0%) | **Trained** (20%) | **Trained** (40%) | **Trained** (60%) | **Trained** (80%) | **Trained** (100%)

Average layer density: **31%**
(69% of activations are 0-valued)