Optimizing Memory Efficiency for Deep Convolutional Neural Networks on GPUs

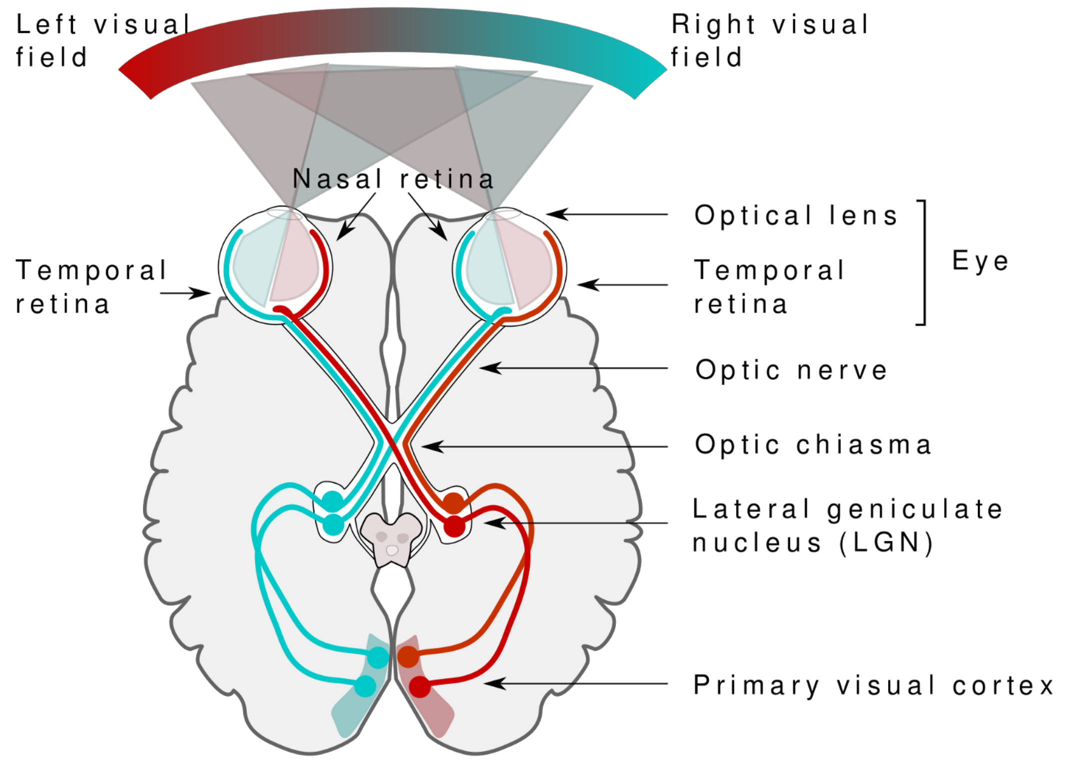
Presenter: Lingqi ZHANG
2018-12-19

Outline

- Background: Convolutional Neural Networks
- Introduction
- Memory Issue A: Data layout
- Memory issue B: off-chip memory accesses
- Results
- Conclusion
- Discussion

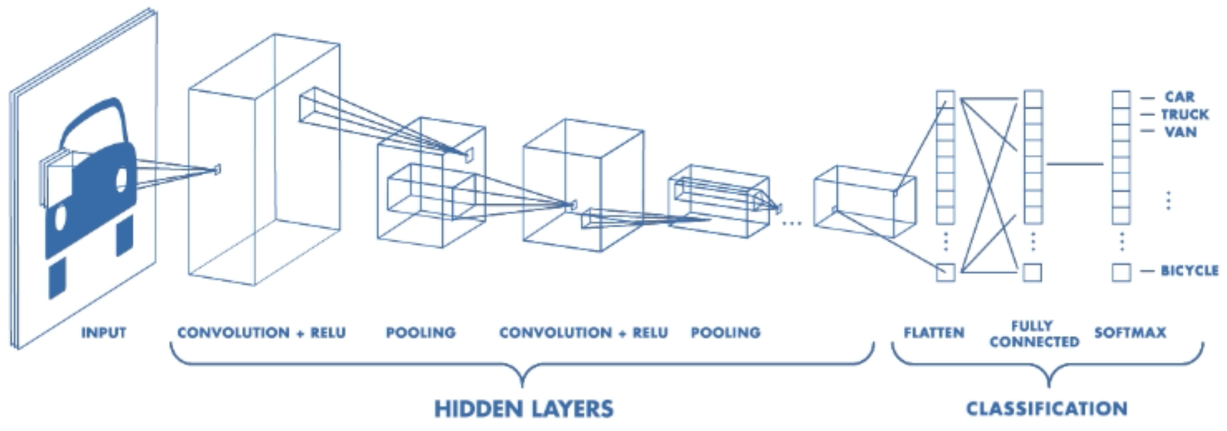
Convolutional Neural Networks (CNN)

- **Visual System[1]**
  - Simple Receptive Field
    - Activation related to location
  - Complex Receptive Field
    - Activation related to patterns



The Visual Pathway. — Source: https://commons.wikimedia.org/wiki/File:Human_visual_pathway.svg

- **CNN[2]**
  - Convolution Layer
  - Pooling Layer
  - Fully Connected Layer
  - Loss Layer
    - Softmax
    - Sigmoid Cross-Entropy
    - Euclidean Loss



Architecture of a CNN. — Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

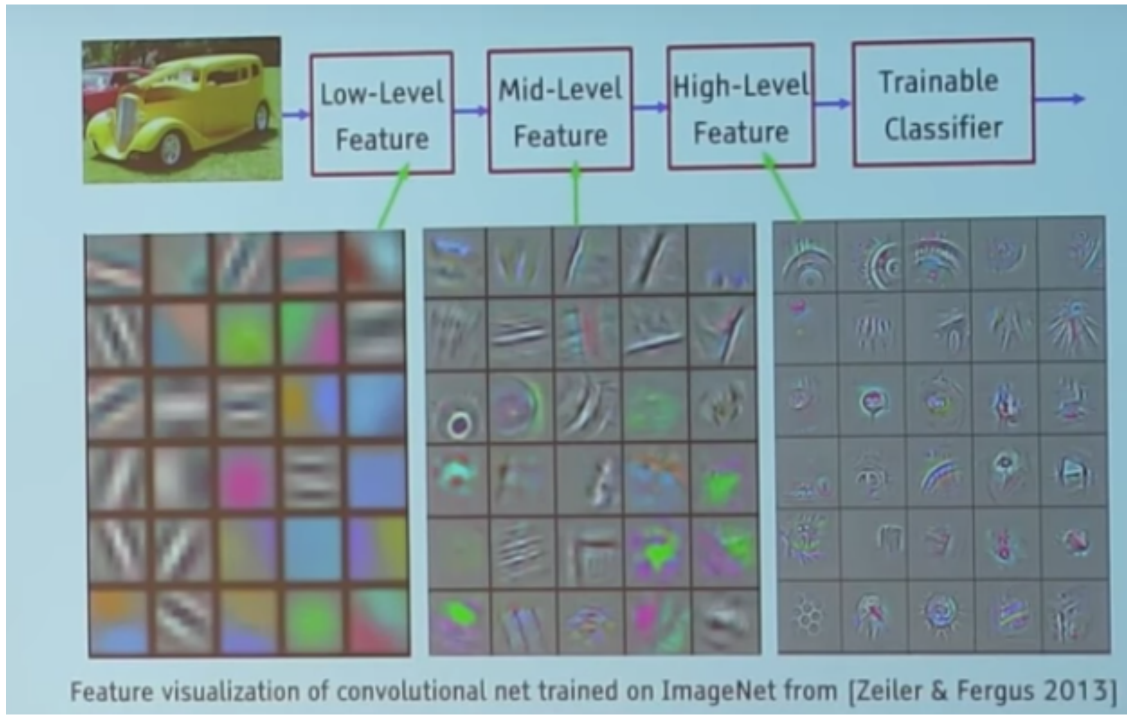
[1] Hubel DH and Wiesel TN. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol, 1962, 160: 106-154 http://jp.physoc.org/content/160/1/106.full.pdf+html
[2] wiki, Convolutional Neural Network, https://en.wikipedia.org/wiki/Convolutional_neural_network

$$Out_{co}[N_i][C_o][H_i][W_i] = \sum_{C_i=0}^{C}\sum_{f_h=0}^{F_H}\sum_{f_w=0}^{F_W} In_{co}[N_i][C_o][H_i+f_h][W_i+f_w] * filter[C_o][C_i][f_h][f_w]$$

Convolutional Layer

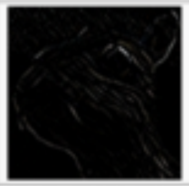
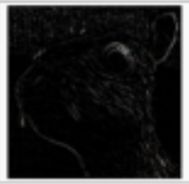
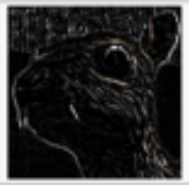
Function:

Extracts features


Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Features in a trained network

https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Different Filters work on Picture

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

$$Out_{po}[N_i][C_o][H_i][W_i] = \sum_{x=0}^{X} \sum_{y=0}^{Y} In_{po}[N_i][C_i][H_i * stride + y][W_i * stride + x]/X/Y$$

Pooling Layer

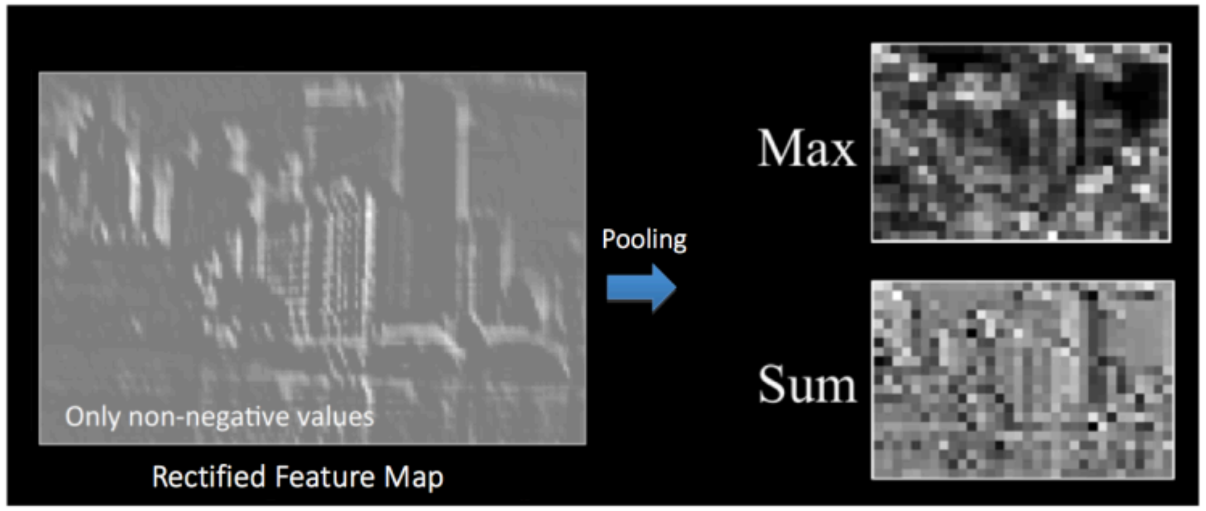
(subsamplint/ downsampling)

Function:

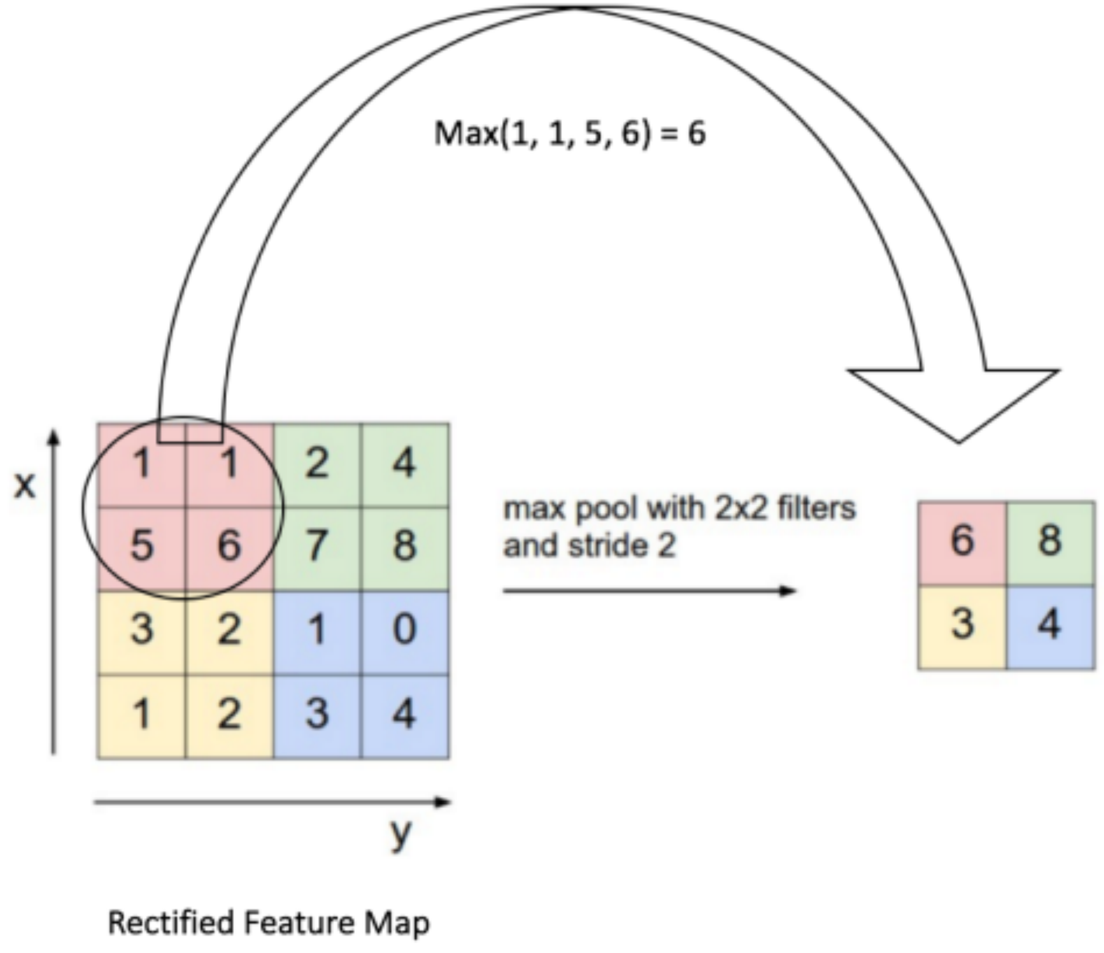
Summarize information of features

Examples:

Max, Ave/Sum



Example of Pooling
http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf



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

Softmax Layer

This paper Specified 'Loss Layer' into 'Softmax Layer'

Function:

User the high level features provided by "previous layer" to do classification.

This paper said that "Before the softmax layer, there usually exist fully-connected layers"
But neglect the discussion of "fully-connected layers" in other parts.

$$Maxv[N_x] = \sum_{x=0}^{X} \sum_{y=0}^{Y} max(In[N_x][C_y])$$

$$Midv1[N_x][C_y] = \sum_{x=0}^{X} \sum_{y=0}^{Y} (In[N_x][C_y] - Maxv[N_x])$$

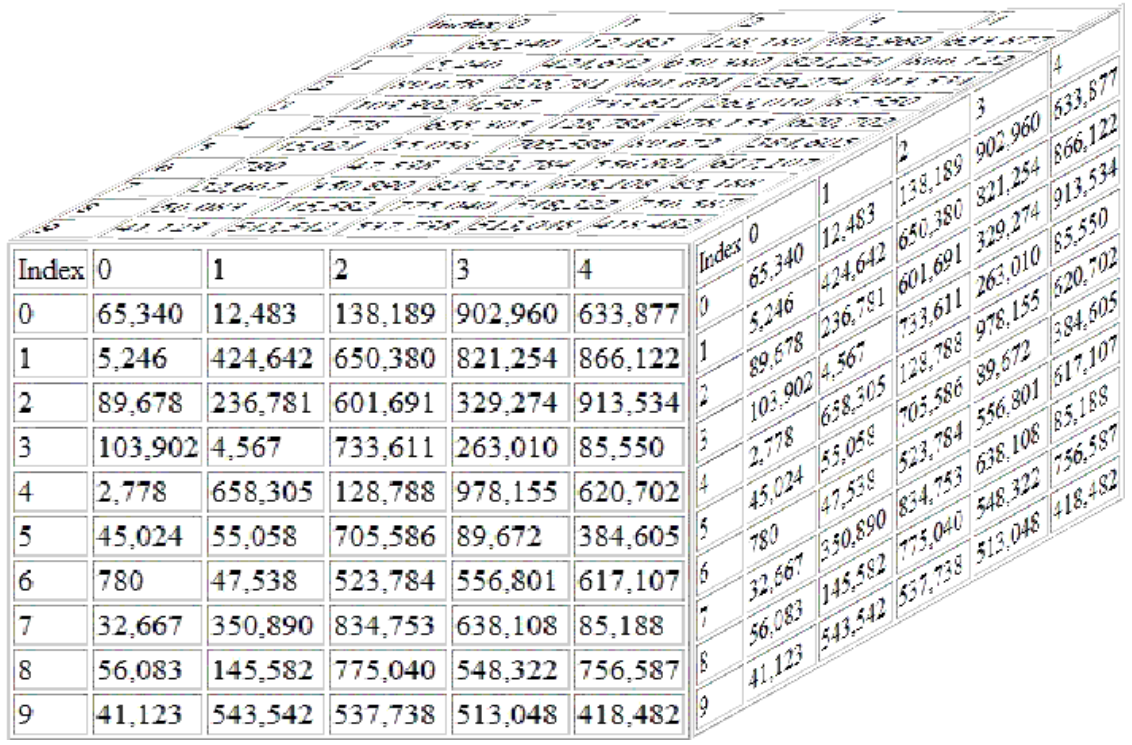
$$Sumv[N_x] = \sum_{x=0}^{X} \sum_{y=0}^{Y} sum(Midv2[N_x][C_y])$$

$$Out[N_x][C_y] = \sum_{x=0}^{X} \sum_{y=0}^{Y} (Midv2[N_x][C_y]/Sumv[N_x])$$

DATA Layout

Definition:
A data layout is a structure applied to a system that defines how the data fields are organized.[1] (First search result by Google)

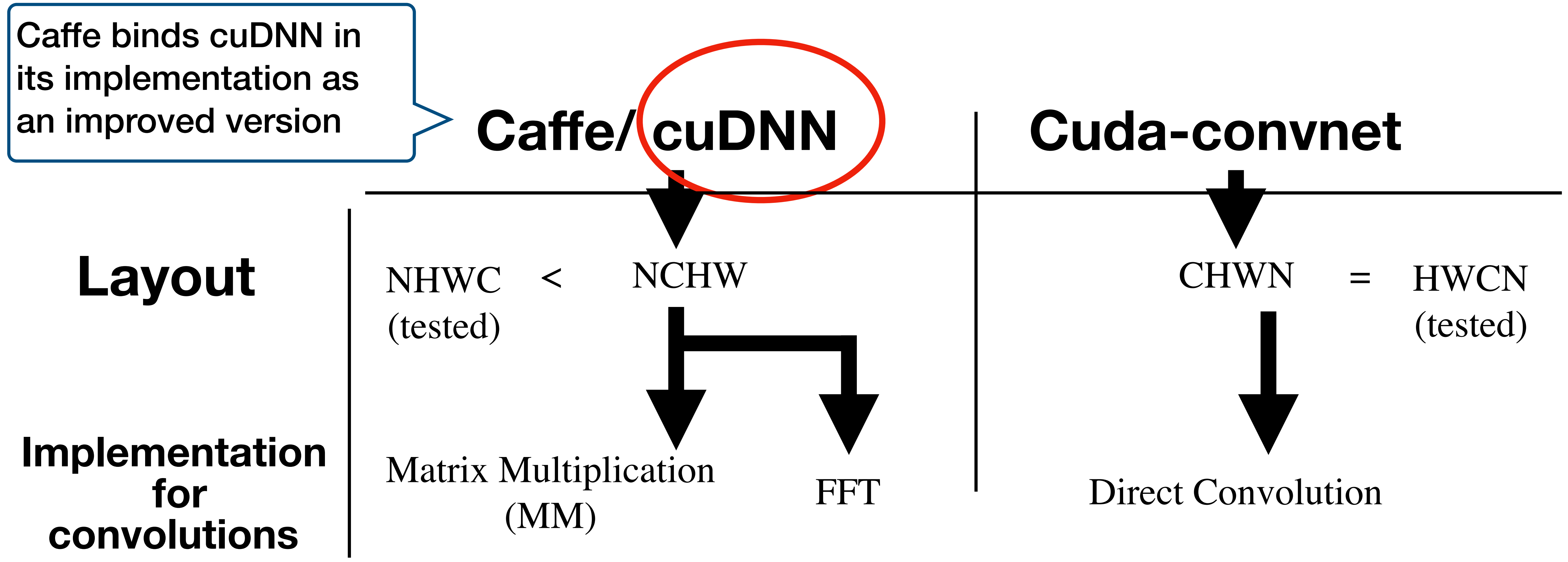
I think the author means the arrangement of multidimensional array.



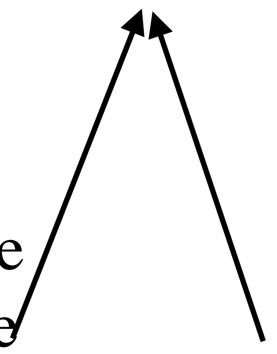
Index	0	1	2	3	4
0	65,340	12,483	138,189	902,960	633,877
1	5,246	424,642	650,380	821,254	866,122
2	89,678	236,781	601,691	329,274	913,534
3	103,902	4,567	733,611	263,010	85,550
4	2,778	658,305	128,788	978,155	620,702
5	45,024	55,058	705,586	89,672	384,605
6	780	47,538	523,784	556,801	617,107
7	32,667	350,890	834,753	638,108	85,188
8	56,083	145,582	775,040	548,322	756,587
9	41,123	543,542	537,738	513,048	418,482

[1]https://help.dsync.com/hc/en-us/articles/115006785467-What-is-a-data-layout-

CNN Libraries


Caffe binds cuDNN in its implementation as an improved version
Caffe/ cuDNN
Cuda-convnet
Layout
NHWC (tested)
<
NCHW
CHWN
=
HWCN (tested)
Implementation for convolutions
Matrix Multiplication (MM)
FFT
Direct Convolution

<u>Nearby index also physically adjacent</u>



N: Batch of image	C: Feature Map
C: Feature Map	H: Height of image
H: Height of image	W: Width of image
W: Width of image	N: Batch of image

Neural Networks

Name	Dataset used when first proposed	Description
LeNet	MINIST	Handwritten character recognition (Number)
Cifar	CIFAR10	10 categories of objects
AlexNet		
ZFNet	ImageNet	1 million real-word images
VGG		

Outline

- Background: Convolutional Neural Networks
- Introduction
- Memory Issue A: Data layout
- Memory issue B: off-chip memory accesses
- Results
- Conclusion
- Discussion

Introduction

- Curent situation
  - Success in CNN (e.g. Alex)
  - GPU optimizations
    - (e.g. Caffe)
  - Reducing arithmetic complexity
- Problems
  - No one focus on memory efficiency
  - 2 issues
    - Data Layouts
    - Redundant off-chip memory access.

Data Layout

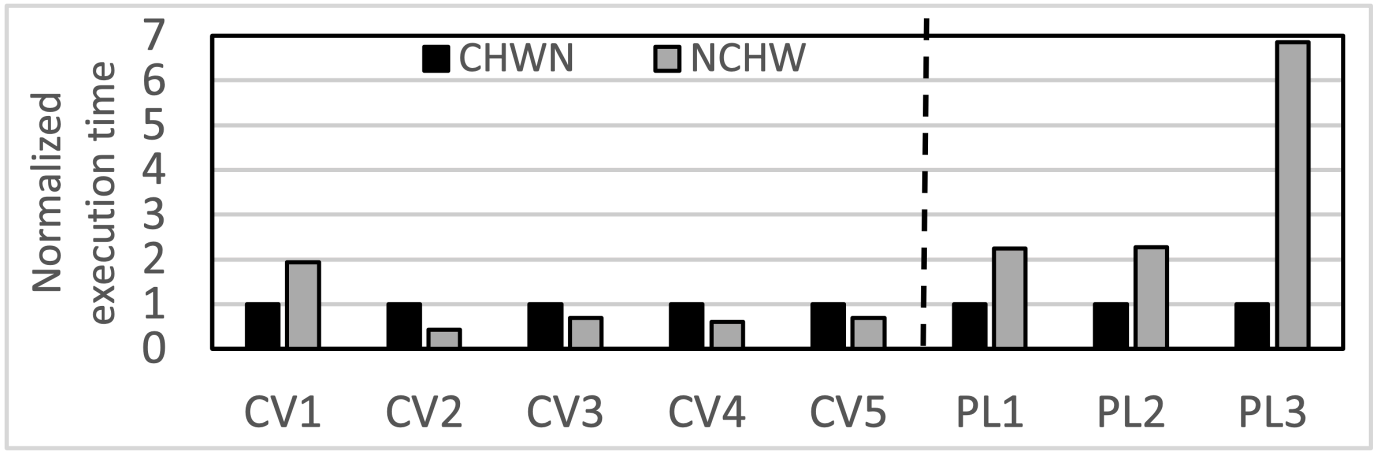


Fig. 1. Performance comparison between the CHWN layout (cuda-convnet2) and NCHW layout (cuDNNv4) on convolutional and pooling layers in AlexNet [12]

- **POINT 1:**
  - GPU thread organization highly depends on data layout
  - Data layout determines the memory access pattern
- **EVIDENCE 1 (Fig1):**
  - Suitable layout lead up to 6.9x layer-level speedup
  - Suitable layout even speedup convolutional layer to up 2.3x
- **POINT 2:** size of each dimension affect performance
  - Because, each dimension has distinct memory access patterns
- **DEDUCTION 1 (from POINT 2):**
  - Performance impact from data layout is complex and difficult to reason about.
- **DEDUCTION 2 (from EVIDENCE 1):**
  - Single data layout cannot deliver the best performance for all the layers.
- **CURRENT SITUATION (PROBLEM):**
  - Current libraries only employ one data layout for all the CNN layers.

Redundant off-chip memory access

- **EVIDENCE (from authors' analysis):**
  - memory-bounded pooling layers and classifier (softmax) layers is far from optimal
  - **DUE TO:** overlook on their off-chip memory data access
- **PROBLEM 1:**
  - CNN requires multiple steps to complete (data dependency exists)
  - **CURRENT SITUATION:** use kernel for each step
  - **PROBLEM:** data pass through the bandwidth-limited off-chip memory
- **PROBLEM 2:**
  - Leveraging data locality for high memory performance is an important optimization
  - **CURRENT SITUATION (PROBLEM):** to optimize locality for different data layouts has not been addressed in existing CNN libraries.

Contributions

Benchmark 1 and optimization 1

1. Benchmarked performance impact of different layouts in various CNN layers. Derived a heuristic guide for layout selection.

Optimization 2

2. Proposed a layout transformation on GPUs. Integrated automatic layout selection and transformation into Caffe

Benchmark 2

3. Benchmarked memory memory behavior of pooling and softmax layers. Further optimize their memory access efficiency on GPUs.

Experiments

4. The authors "perform rigorous evaluation and result analysis on different types of layers and representative networks, and demonstrate high performance improvements for both single layers, and complete networks"

I think the authors want to express that they applied their optimizations in different types of layers and representative networks.

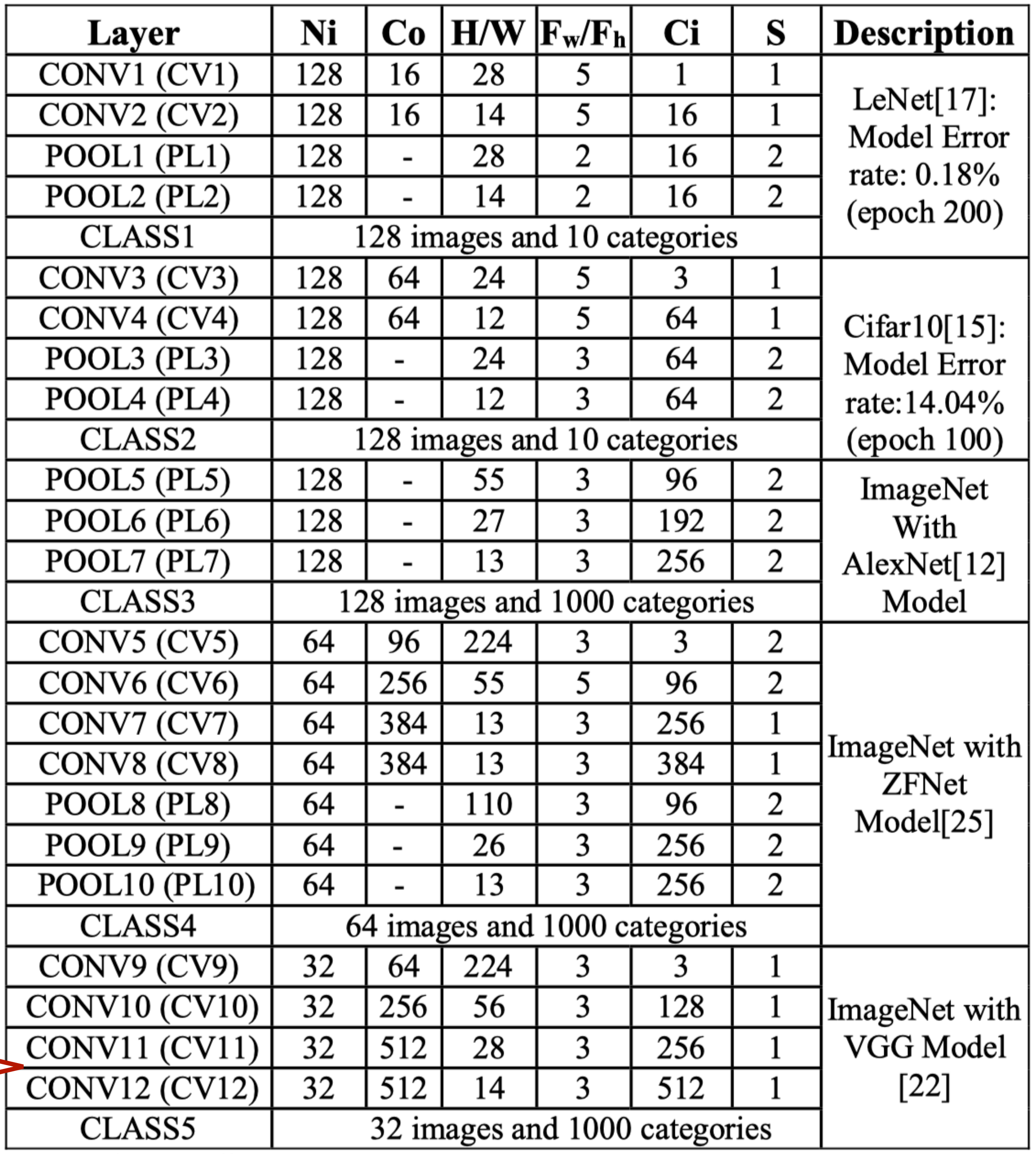
Benchmarks

- N: Batch
- C: Feature Map
- H/W: Image size
- Fh/Fw: Filter size
- S: slide

- **LAYERS:**
  - TABLE 1 shows the layers chosen from famous neural networks.

  - Convolutional layer comes from this table.
  - Pooling layer comes from this table.
  - Softmax layer is benchmarked by several settings (described in section VI).

It's interesting that this paper did not benchmark convolutional layer in Alex Nex and pooling layer in VGG



Layer	Ni	Co	H/W	F_w/F_h	Ci	S	Description
CONV1 (CV1)	128	16	28	5	1	1	LeNet[17]: Model Error rate: 0.18% (epoch 200)
CONV2 (CV2)	128	16	14	5	16	1	
POOL1 (PL1)	128	-	28	2	16	2	
POOL2 (PL2)	128	-	14	2	16	2	
CLASS1	128 images and 10 categories						
CONV3 (CV3)	128	64	24	5	3	1	Cifar10[15]: Model Error rate:14.04% (epoch 100)
CONV4 (CV4)	128	64	12	5	64	1	
POOL3 (PL3)	128	-	24	3	64	2	
POOL4 (PL4)	128	-	12	3	64	2	
CLASS2	128 images and 10 categories						
POOL5 (PL5)	128	-	55	3	96	2	ImageNet With AlexNet[12] Model
POOL6 (PL6)	128	-	27	3	192	2	
POOL7 (PL7)	128	-	13	3	256	2	
CLASS3	128 images and 1000 categories						
CONV5 (CV5)	64	96	224	3	3	2	ImageNet with ZFNet Model[25]
CONV6 (CV6)	64	256	55	5	96	2	
CONV7 (CV7)	64	384	13	3	256	1	
CONV8 (CV8)	64	384	13	3	384	1	
POOL8 (PL8)	64	-	110	3	96	2	
POOL9 (PL9)	64	-	26	3	256	2	
POOL10 (PL10)	64	-	13	3	256	2	
CLASS4	64 images and 1000 categories						
CONV9 (CV9)	32	64	224	3	3	1	ImageNet with VGG Model [22]
CONV10 (CV10)	32	256	56	3	128	1	
CONV11 (CV11)	32	512	28	3	256	1	
CONV12 (CV12)	32	512	14	3	512	1	
CLASS5	32 images and 1000 categories						

TABLE 1: THE CNNS AND THEIR LAYERS USED IN THE EXPERIMENTS.

Outline

- Background: Convolutional Neural Networks
- Introduction
- Memory Issue A: Data layout
- Memory issue B: off-chip memory accesses
- Results
- Conclusion
- Discussion

A
Data Layout in Convolutional Layers

I think these authors mean “Direct Convolution" and “Matrix Multiplication” here

- **Benchmark:**
  - Comparison of CHWN and NCHW with their best performance implementation (cuda-convnet and cuDNN respectively)
- **Observations:**
  - cuda_convnet outperform cuDNN for CV1-5 and CV9
    - Because C no larger than 64

I think these authors mean Ci here

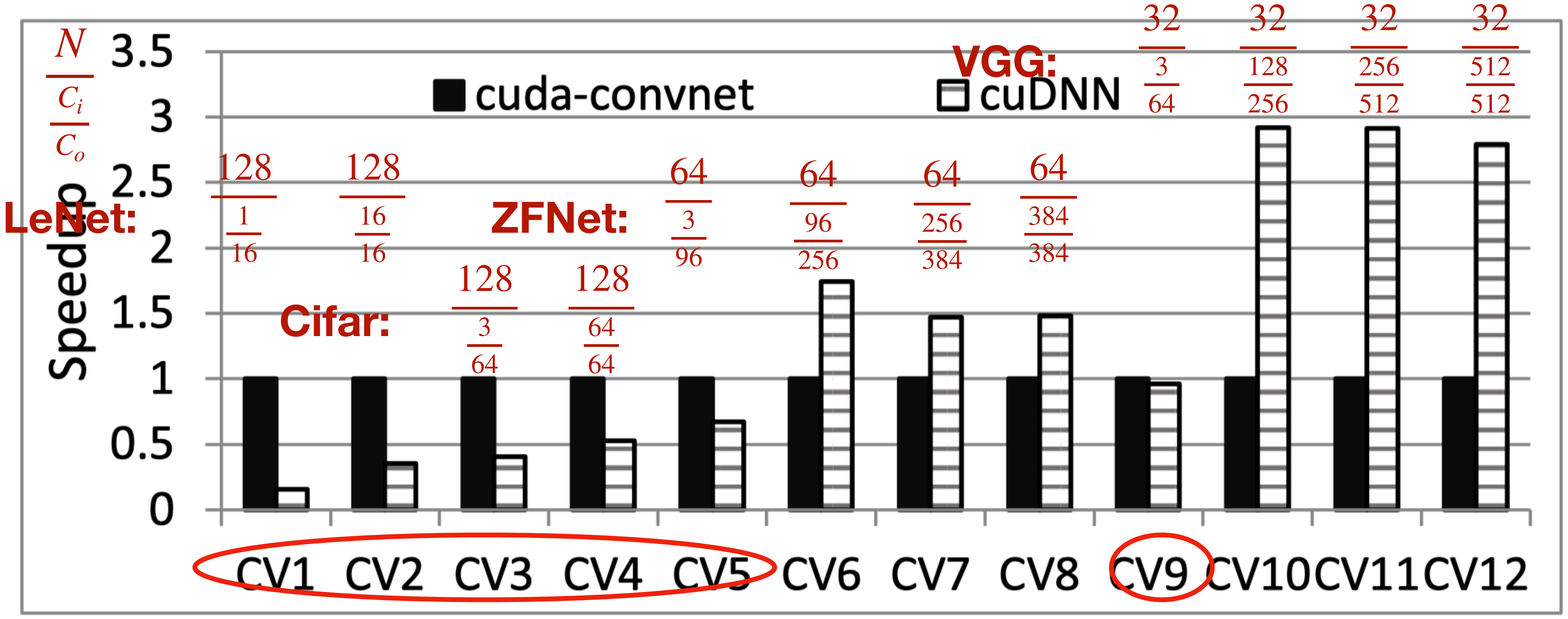


Fig. 3. Performance comparison between two different data layouts for the convolutional layers in Table 1. The performance is normalized to cuda-convnet measured on a GTX TITAN BLACK.

A
Data Layout in Convolutional Layers

Again, Ci here

- **Benchmark:**
  - To further identify the sensitivities of data layouts on each dimension, the researchers collect the results with one varying dimension size (N or C) and the other three being fixed

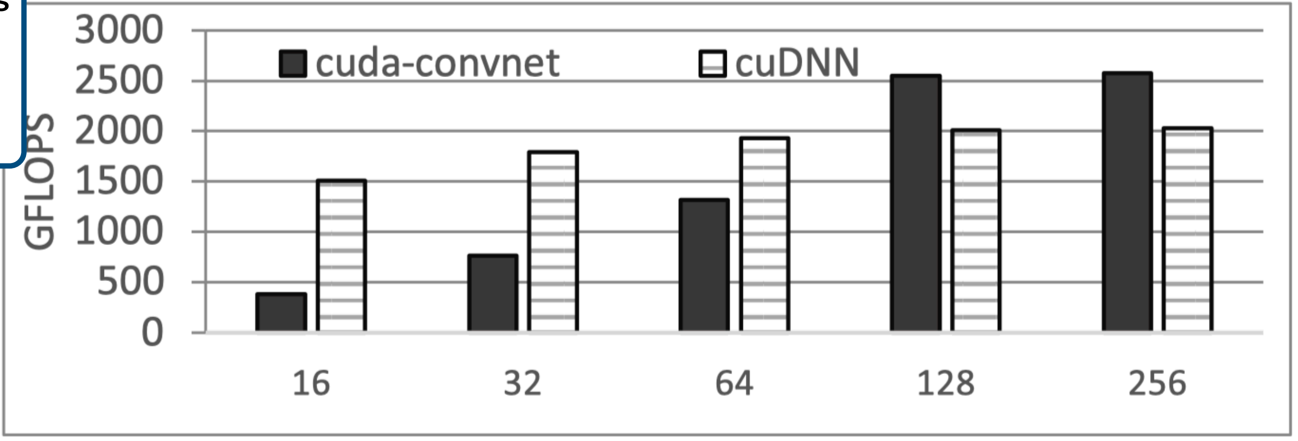
If batch size N is 128, cuda-convnet enables each thread to handle 4 images.
If batch size is less, the reuse for images per thread would be reduces

- **Analysis:**
  - if N<128 :
    - Cuda-convnet could not achieve top performance (implementation related)
  - If C < 32:
    - Overhead of unrolling matrix (in cuDNN) is more evidence

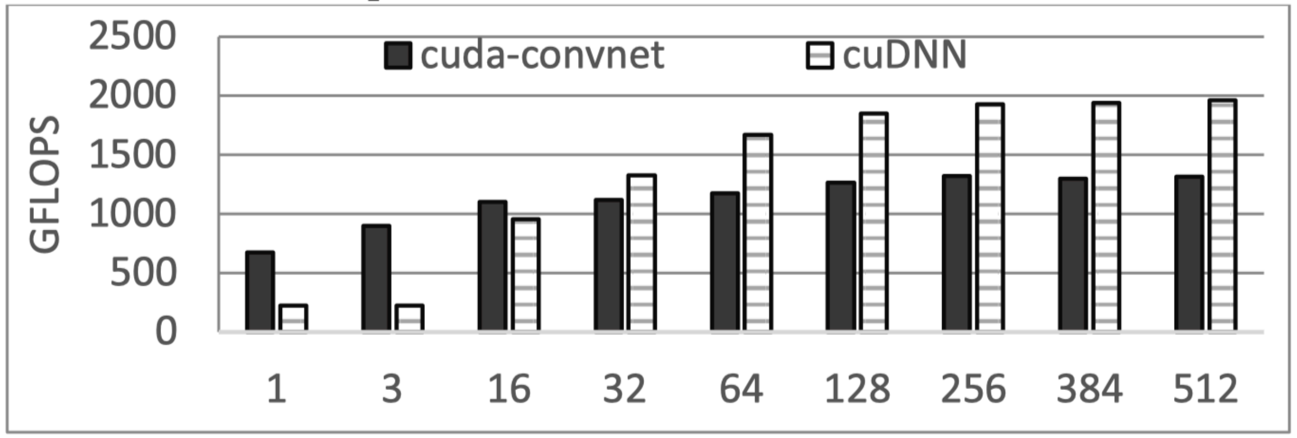
Significant?

- **Heuristic Layout Selection:**
  - if C<Ct or N>Nt :
    - Better to choose CHWN
  - Else
    - Better to choose NCHW

Varies depends one system.
Titan Black:
(Ct=32, Nt=128)
Titan X:
(Ct=128, Nt=64)



a. The performance with different values of N.



b. The performance with different values of C.

Fig. 4. Sensitivity study of data layouts on the N and C dimensions. CONV7 in Table 1 is used while others show similar trends.

A
Data Layout in Convolutional Layers

- **Benchmark:**
  - Performance of various convolutional layers using FFT, FFT-Tiling and Matrix Multiplication with the NCHW layout compared to cuda-convenet with the CHWN data layout.
- **Observation:**
  - If {N is large}||{filter kernel is large}||{C is large}
    - FFT is better than MM
    - **REASON:**
      - overhead in forward and backword FFT.
  - Heuristic Layout selection still works

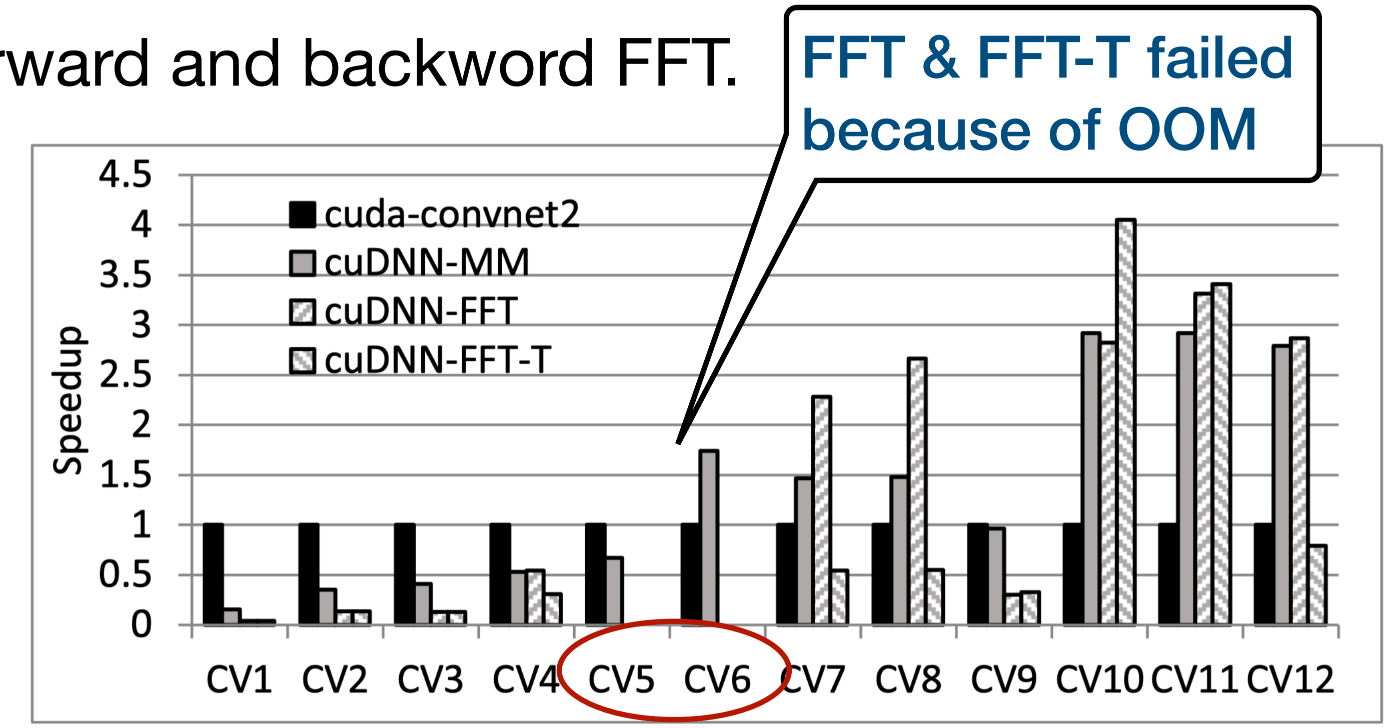


Fig. 5. Speedups of the FFT-based approach over the cuda-convets.

B
Data Layout in Pooling Layers

- **Benchmark:**
  - Performance of pooling layers with different data layouts
  - Cuda-convnet (CHWN) vs Caffe & cuDNN (NCHW)
- **Conclusion:**
  - CHWN always better than NCHW
  - **REASON:**
    - NCHW layout cannot ensure coalesced memory access.

Theoretical Peak: 235GB/s

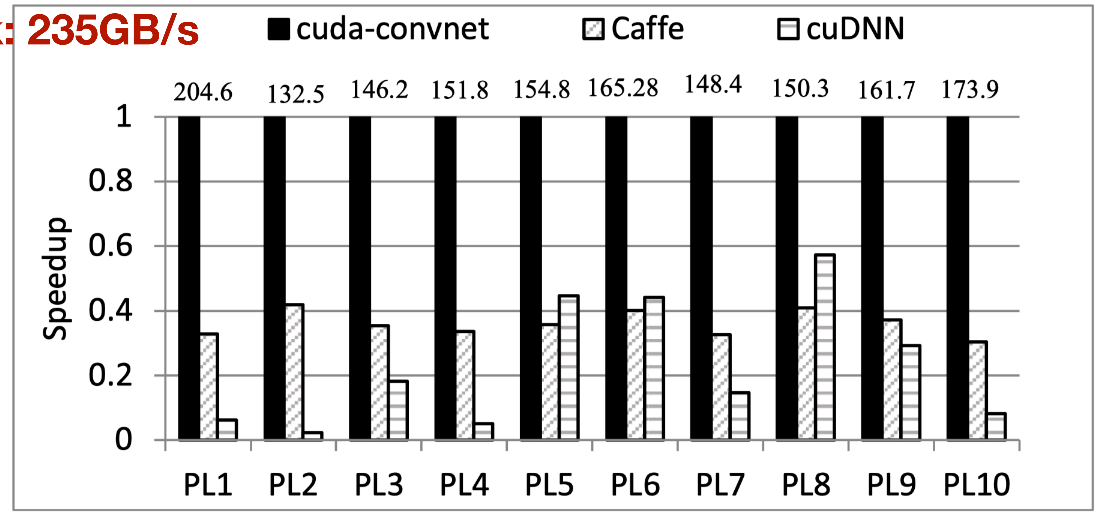


Figure 6. Performance comparison between different data layouts for the pooling layers in Table 1. The performance is normalized to cuda-convnet. The numbers on top denote the highest bandwidth (GB/S) achieved for each layer.

C
A Fast Data Layout Transformation for CNNs

In fact CHWN->NCHW

<16/32,32>,<H,W> According to the result of C%32

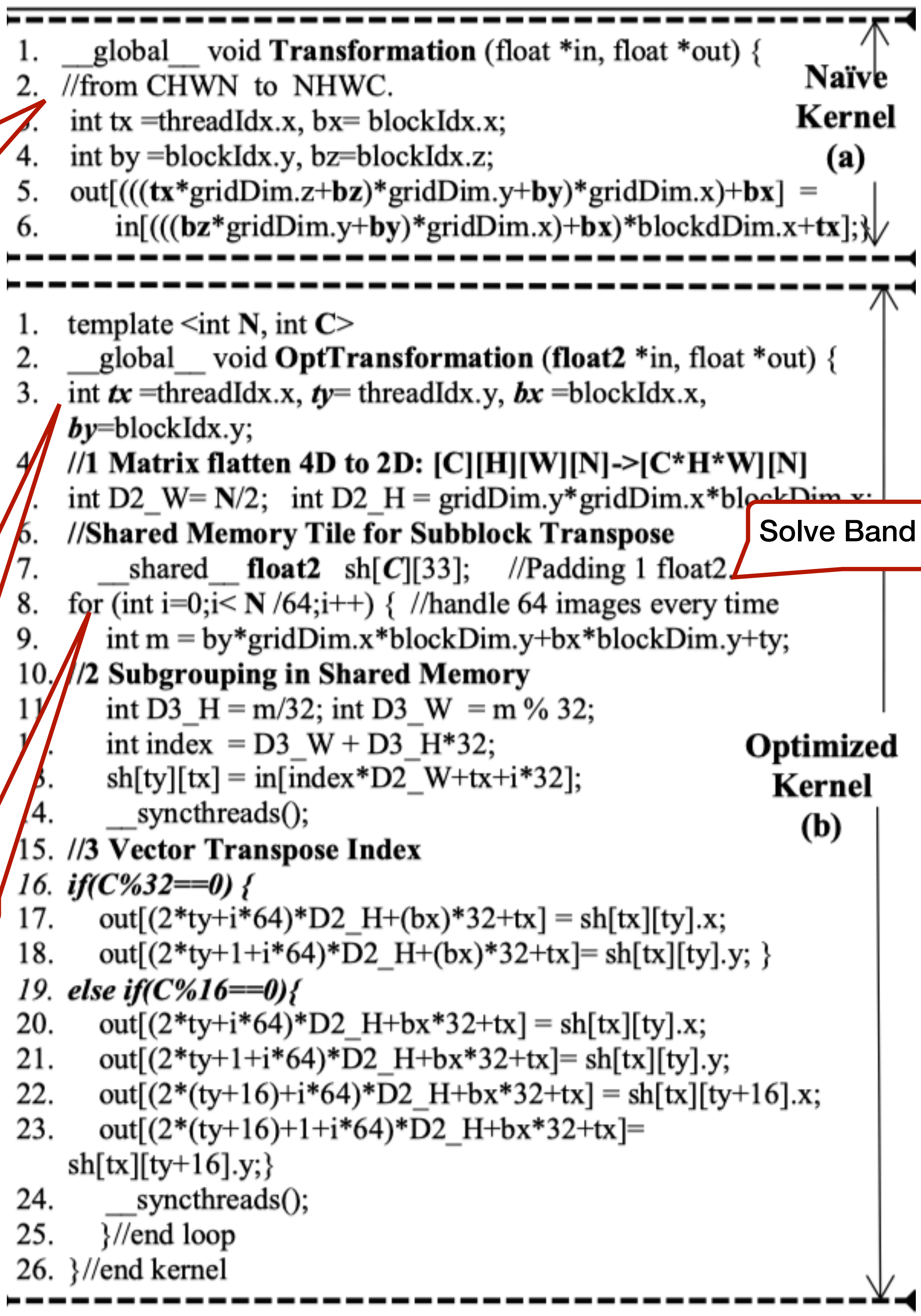
Iterate to all Batches

Solve Band Conflict

Optimizations:

1. Change 4D transform to 2D transform
2. Shared memory tiling
3. Vectorization to use 8 byte access

I think vectorization increase the performance by increase bandwidth of global memory here



```
__global__ void Transformation (float *in, float *out) {
//from CHWN to NHWC.
 int tx =threadIdx.x, bx= blockIdx.x;
 int by =blockIdx.y, bz=blockIdx.z;
 out[(((tx*gridDim.z+bz)*gridDim.y+by)*gridDim.x)+bx] =
     in[(((bz*gridDim.y+by)*gridDim.x)+bx)*blockdDim.x+tx];}
```

Naïve Kernel (a)

```
template <int N, int C>
__global__ void OptTransformation (float2 *in, float *out) {
int tx =threadIdx.x, ty= threadIdx.y, bx =blockIdx.x,
by=blockIdx.y;
//1 Matrix flatten 4D to 2D: [C][H][W][N]->[C*H*W][N]
int D2_W= N/2;  int D2_H = gridDim.y*gridDim.x*blockDim.x;
//Shared Memory Tile for Subblock Transpose
   __shared__ float2  sh[C][33];    //Padding 1 float2.
for (int i=0;i< N /64;i++) {  //handle 64 images every time
   int m = by*gridDim.x*blockDim.y+bx*blockDim.y+ty;
//2 Subgrouping in Shared Memory
   int D3_H = m/32; int D3_W  = m % 32;
   int index  = D3_W + D3_H*32;
   sh[ty][tx] = in[index*D2_W+tx+i*32];
     __syncthreads();
//3 Vector Transpose Index
if(C%32==0) {
   out[(2*ty+i*64)*D2_H+(bx)*32+tx] = sh[tx][ty].x;
   out[(2*ty+1+i*64)*D2_H+(bx)*32+tx]= sh[tx][ty].y; }
else if(C%16==0){
   out[(2*ty+i*64)*D2_H+bx*32+tx] = sh[tx][ty].x;
   out[(2*ty+1+i*64)*D2_H+bx*32+tx]= sh[tx][ty].y;
   out[(2*(ty+16)+i*64)*D2_H+bx*32+tx] = sh[tx][ty+16].x;
   out[(2*(ty+16)+1+i*64)*D2_H+bx*32+tx]=
sh[tx][ty+16].y;}
     __syncthreads();
   }//end loop
}//end kernel
```

Fig. 7. Kernel code for data layout transformation

D
Wrap Up: Automatic CNN Data Layout Support

- Code Modification:
  - add a new field in each convolutional and pooling layer to indicate the data layout choice.
    - Use the heuristic method proposed to set layout.
  - At the completion time of layer, an additional check is needed, to determine the overhead of data layout transformation over the performance improvement obtained from the suitable data layout.

Outline

- **Background: Convolutional Neural Networks**
- **Introduction**
- **Memory Issue A: Data layout**
- **Memory issue B: off-chip memory accesses**
- **Results**
- **Conclusion**
- **Discussion**

A
Memory Analysis and Optimization on Pooling Layers

- **ANOTHER PROBLEM:**
  - Redundant data access
- **SOLUTION:**
  - THREAD FUSING
    - Cache the input data in Register File for reuse

Theoretical Peak: 235GB/s

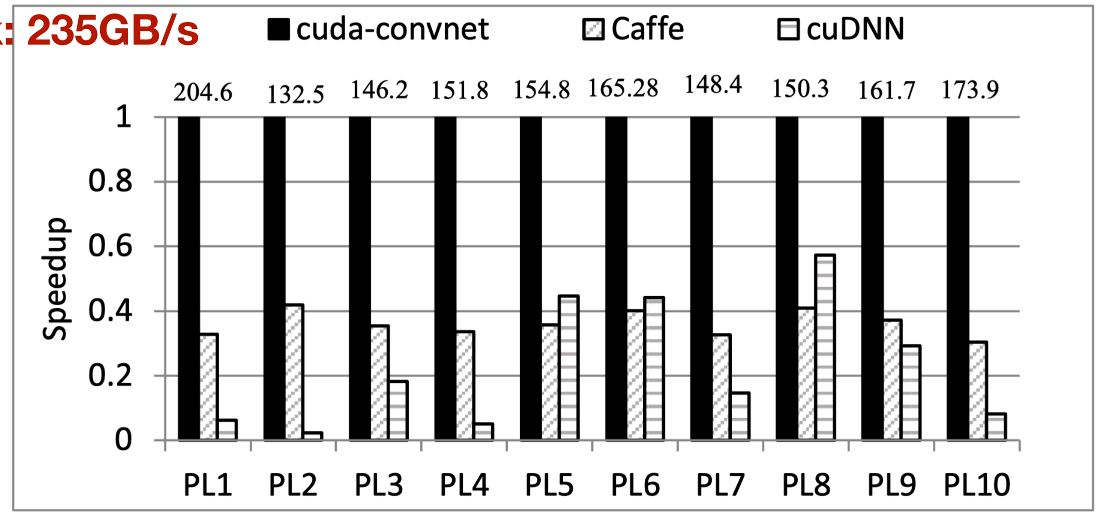


Figure 6. Performance comparison between different data layouts for the pooling layers in Table 1. The performance is normalized to cuda-convnet. The numbers on top denote the highest bandwidth (GB/S) achieved for each layer.

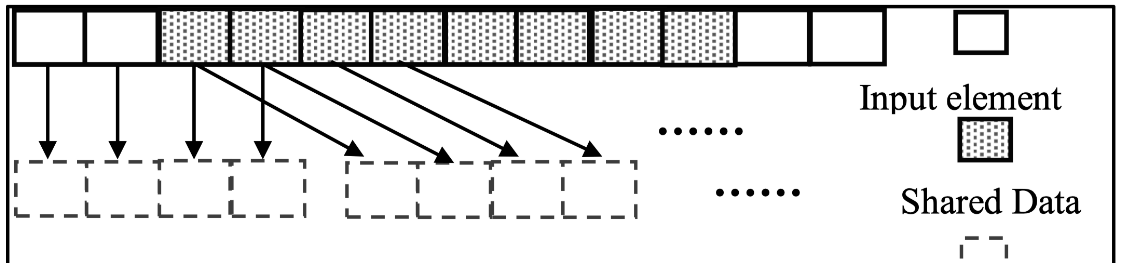


Fig. 8. Overlapped pooling with a window size of 4. The 2D image is simplified to 1D vector.

B
Memory Analysis and Optimization on Softmax Layers

$$Maxv[N_x] = \sum_{x=0}^{X}\sum_{y=0}^{Y} max(In[N_x][C_y])$$

$$Midv1[N_x][C_y] = \sum_{x=0}^{X}\sum_{y=0}^{Y} (In[N_x][C_y] - Maxv[N_x])$$

$$Sumv[N_x] = \sum_{x=0}^{X}\sum_{y=0}^{Y} sum(Midv2[N_x][C_y])$$

$$Out[N_x][C_y] = \sum_{x=0}^{X}\sum_{y=0}^{Y} (Midv2[N_x][C_y]/Sumv[N_x])$$

- **PROBLEM:**
  - The highest bandwidth achieved for the softmax layers (BL_Best) is far from optimization
- **ANALYSIS:**
  - There are 5 kernels to compute softmax layers step by step, which involves redundancy in using global memory.
  - Not enough parallelism in inner loop
- **SOLUTION:**
  - Kernel fusing
  - Inter-step communication with share memory
  - Parallel inner loop

Theoretical Peak: 235GB/s

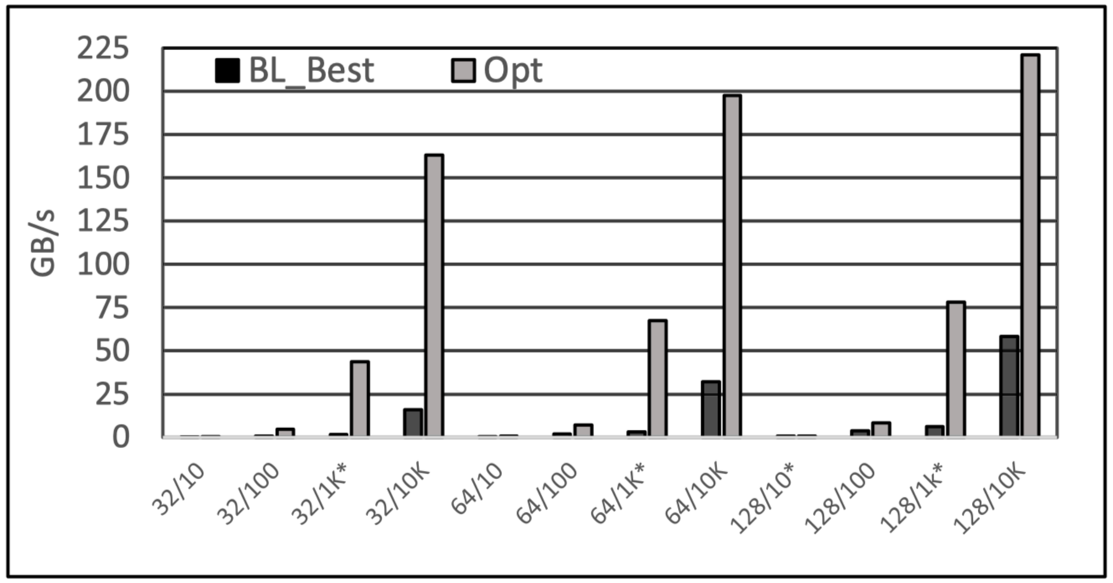
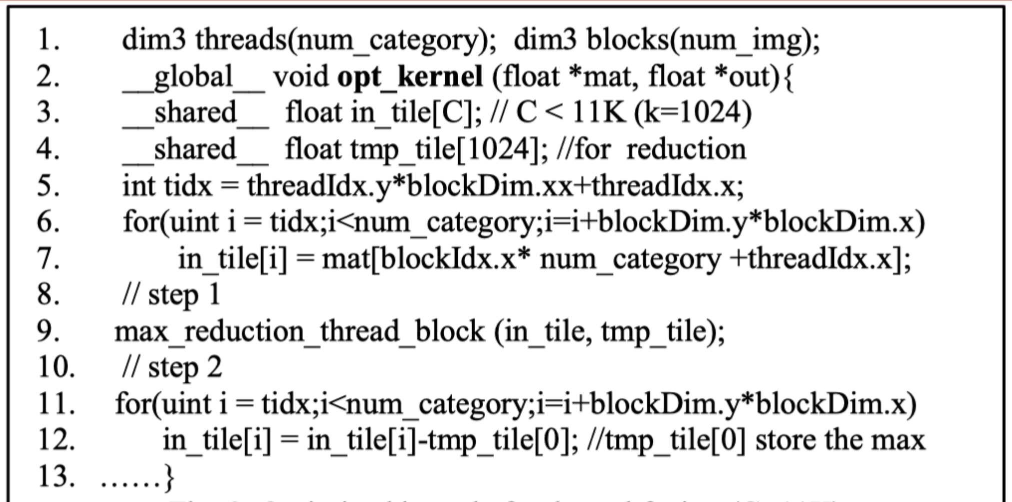


Fig. 13. Performance comparison (GB/S) of softmax layers with a wide range of configurations. x/y means the batch size as x and the number of categories as y.

I guess here means the computation of a window with size X x Y



```
dim3 threads(num_category);  dim3 blocks(num_img);
__global__ void opt_kernel (float *mat, float *out){
__shared__  float in_tile[C]; // C < 11K (k=1024)
__shared__  float tmp_tile[1024]; //for  reduction
int tidx = threadIdx.y*blockDim.xx+threadIdx.x;
for(uint i = tidx;i<num_category;i=i+blockDim.y*blockDim.x)
      in_tile[i] = mat[blockIdx.x* num_category +threadIdx.x];
// step 1
max_reduction_thread_block (in_tile, tmp_tile);
// step 2
for(uint i = tidx;i<num_category;i=i+blockDim.y*blockDim.x)
     in_tile[i] = in_tile[i]-tmp_tile[0]; //tmp_tile[0] store the max
……}
```

Fig. 9. Optimized kernel after kernel fusion (C<11K)

Outline

- Background: Convolutional Neural Networks
- Introduction
- Memory Issue A: Data layout
- Memory issue B: off-chip memory accesses
- Results
- Conclusion
- Discussion

Speedup of a layout over an alternative one

A
Results on Data Layout Optimization

- **Experiment:**
  - Performance of transformation
  - overhead of transformation when ensemble it into Convolutional Layer
- **CONCLUSION:**
  - Data layout has significant performance impact
  - Optimizations in transformation works
  - By considering the data layout transformation overhead, Most of layers still gain performance by doing transformation in layout.

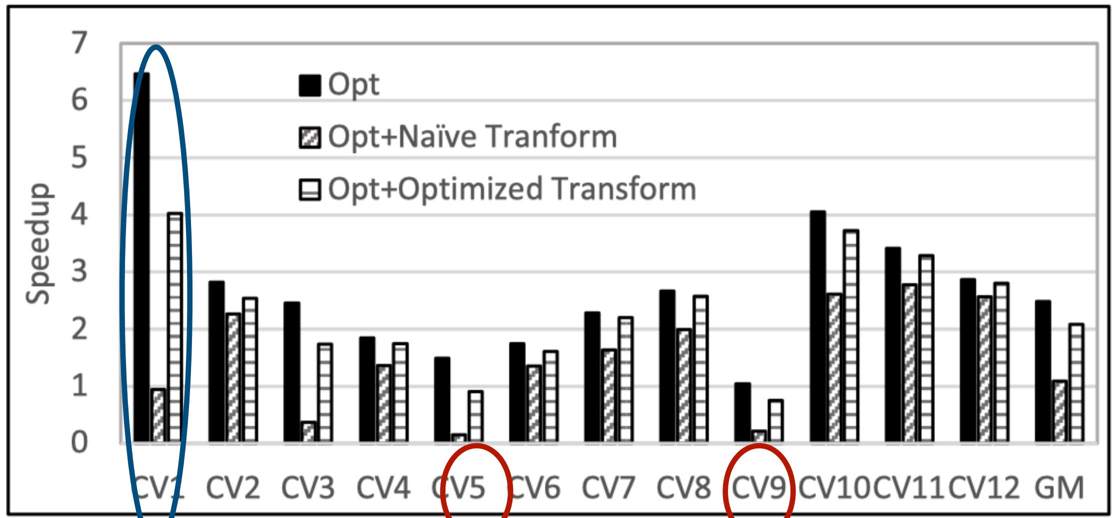


Fig. 10. Speedups achieved on all convolutional layers. For both NCHW and CHWN data layouts, the best achieved performance is measured to calculate the performance differences.

Theoretical Peak: 235GB/s

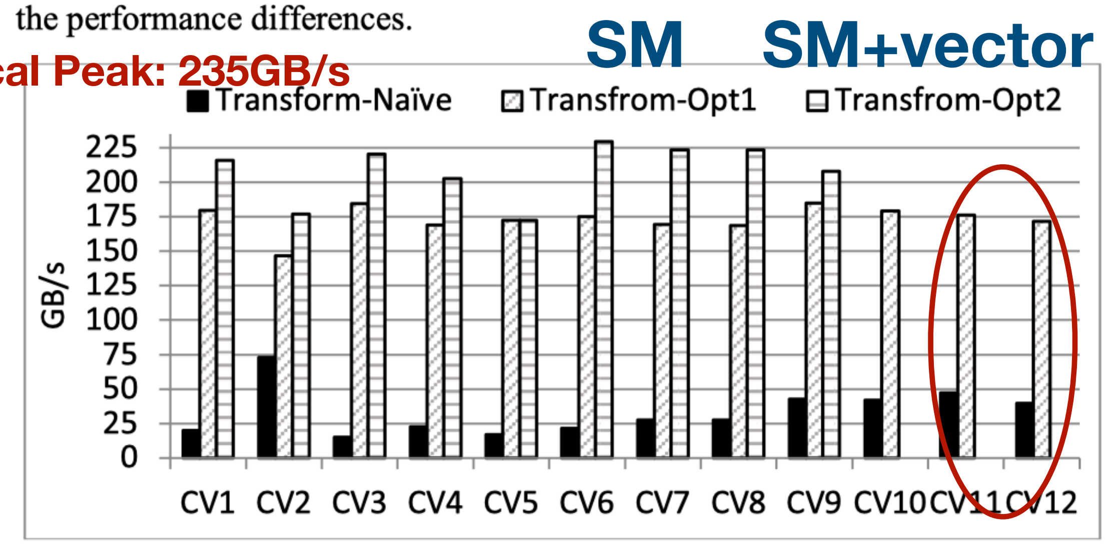


Fig. 11. Achieved memory bandwidth using three methods for data layout transformation. The Transform-Opt2 is not applicable for CV10, CV11, CV12

with an average of 7.5x speedup
Memory overhead: 73.5MB

REF

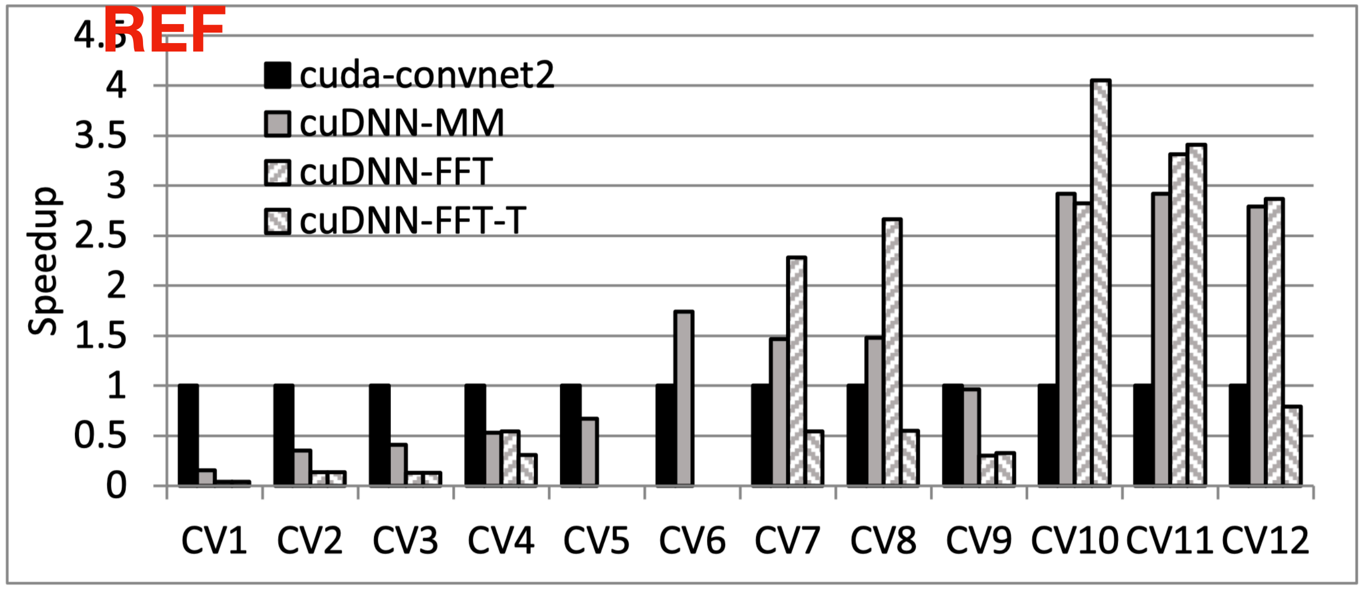


Fig. 5. Speedups of the FFT-based approach over the cuda-convets.

B
Results on Off-chip Memory Access Optimization

- **Experiments:**
  - Performance comparison of different pooling layers
  - Memory bandwidth comparison between optimized and original best alternative implementation.

reduced 9.1% global memory transactions
36% DRAM accesses
Achieve higher performance with an average of 14.3%

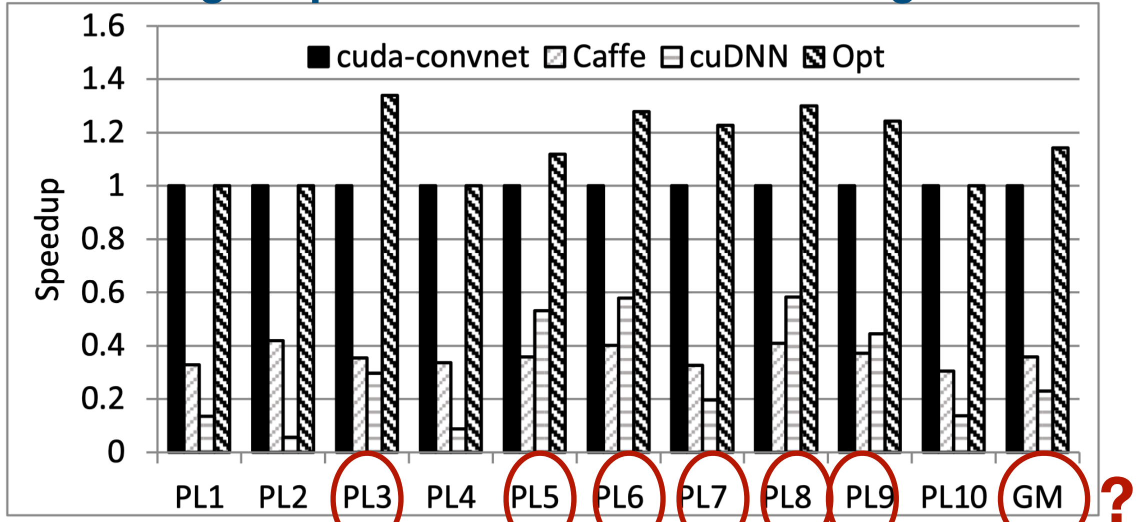


Fig. 12. Performance comparison among four different implementations for the pooling layers in Table 1. The performance is normalized to cuda-convnet.

Theoretical Peak: 235GB/s

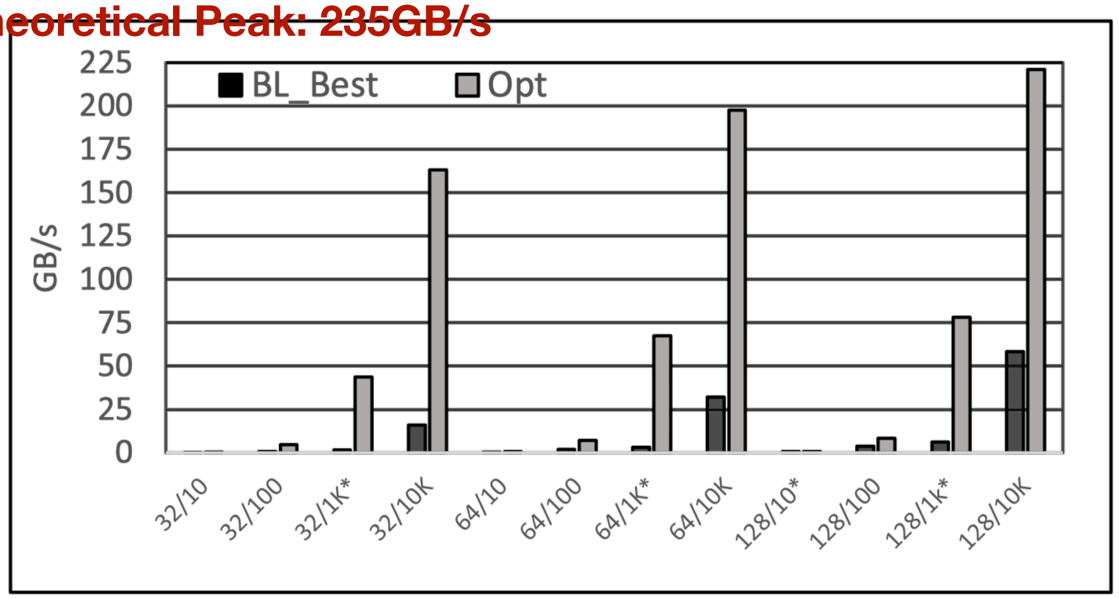


Fig. 13. Performance comparison (GB/S) of softmax layers with a wide range of configurations. x/y means the batch size as x and the number of categories as y.

Communication: 2.81x speedup average
Parallel inner loop: 5.13x speedup average

B
Results on Whole Networks

- Experiments:
  - Integrate optimizations into cuDNN and compare.
  - detailed performance comparison of different layers in AlexNet.
- Conclusion:
  - Flexible data layout: 72% improvement
  - Off-chip memory access optimization contributes 28%

- Pooling layers:
  - speedup by 27.8%
- Softmax layers:
  - 20.1x over cuDNN
  - 8.2x over cuda-convnet
- Overall
  - 16% over cuda-convnet
  - 46% over cuDNN-Best

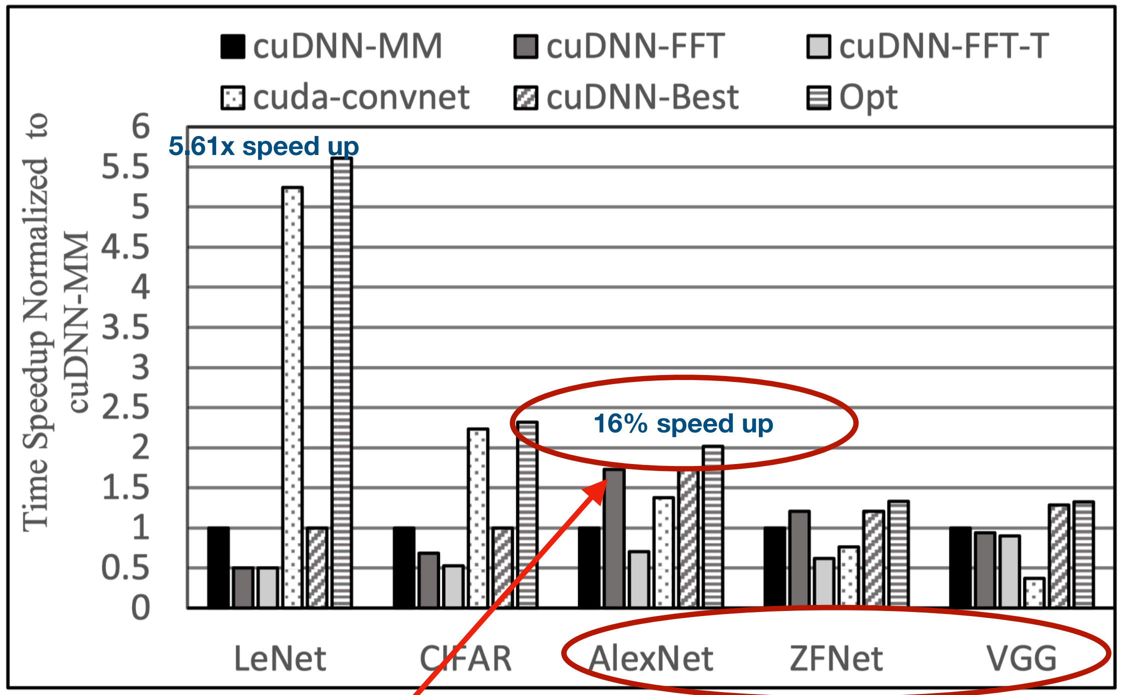


Fig. 14. The overall network performance comparison among various schemes.

ImageNet group

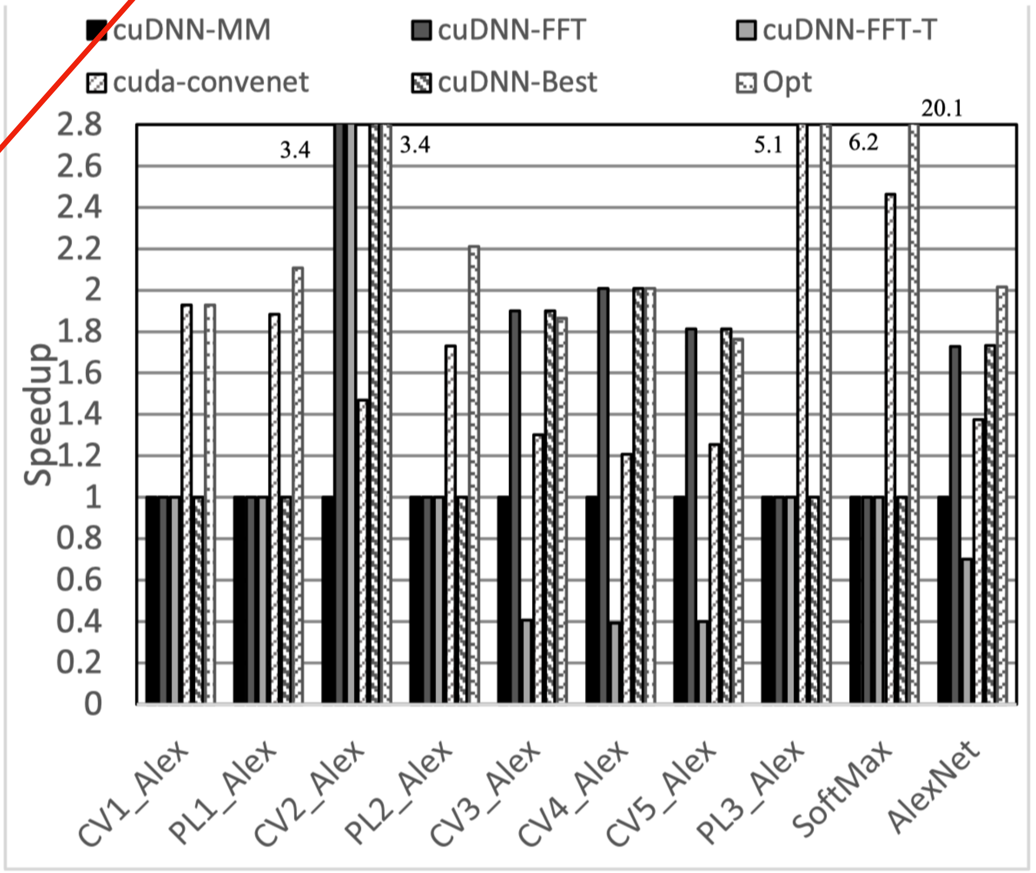


Fig. 15. The performance comparison of different layers in AlexNet, The performance is normalized to cuDNN-MM.

Outline

- Background: Convolutional Neural Networks
- Introduction
- Memory Issue A: Data layout
- Memory issue B: off-chip memory accesses
- Results
- Conclusion
- Discussion

CONCLUSION(cite)

- “Our detailed study unveils the impact of data layouts on different types of CNN layers and their performance implications.”
- “We propose efficient data layout support as our solution.”
- “We further look into the memory access patterns of the memory-bounded layers, and propose effective optimizations to substantially reduce their off-chip memory requests and inter-kernel communication.”

Outline

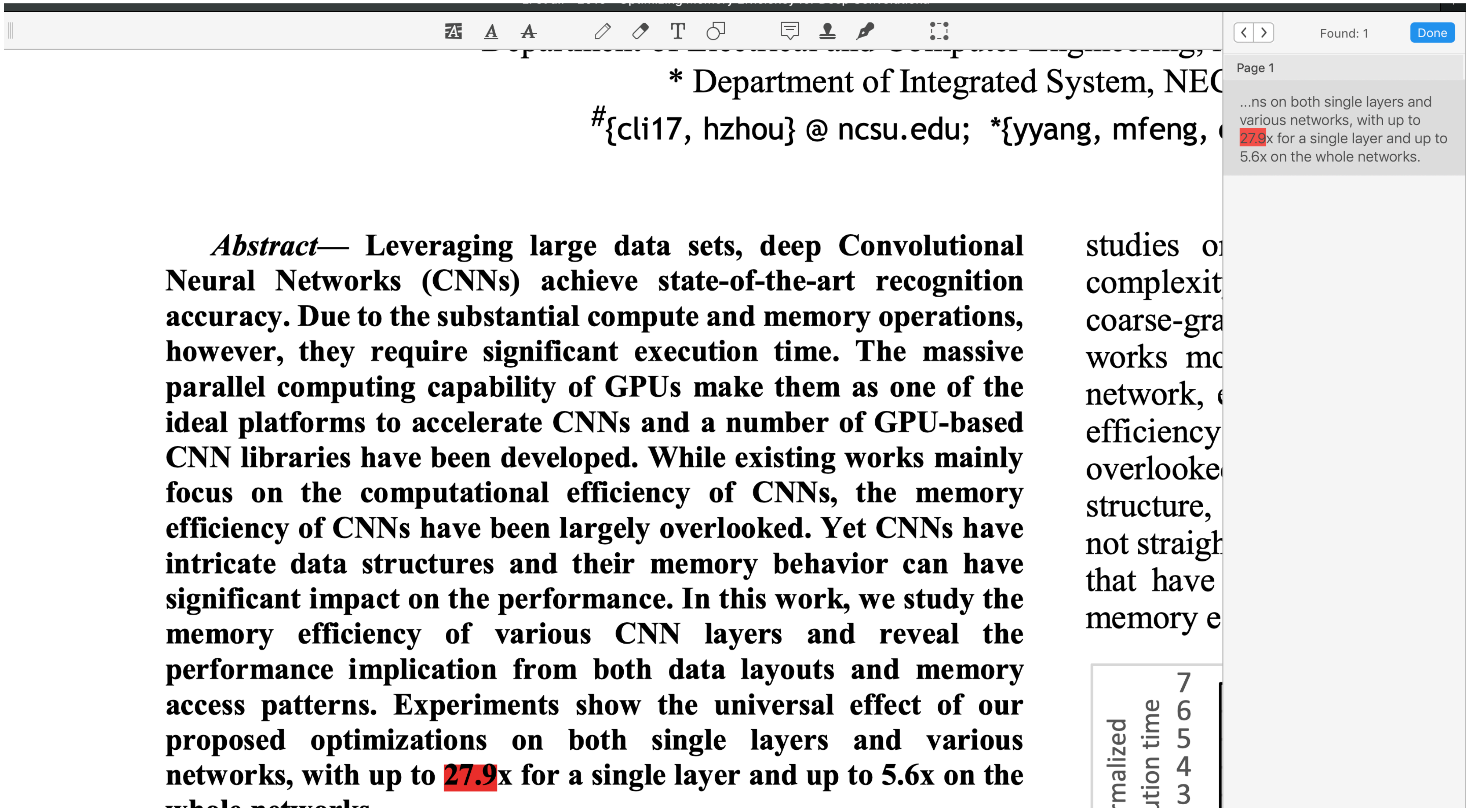
Discussion

- **Things I considered as big issues.**
  - Experiment setting without carefully control variable
  - Changing Concepts several times
  - Neglect the fact that Pooling layers (better to use NCHW) are nearly always inserted into two Convolutional Layers (CHWN). (Lack experiments)
  - Over-exaggerate
- **Things I considered as misses.**
  - Inconsistance
  - Neglect ReLU Layer and Fully Connected Layer.
  - CODEs not present well
  - Never mention time consumption
- **Things could be done better.**
  - Further experiment to compare the performance difference between different convolutional implementations (with same layout)
  - Provide analysis of Cache Miss in Pooling Layers.
  - Explain why specific Layer is chosen while others not

Change Concepts

- NCHW = best implementation in NCHW layout = cuDNN MM
- CHWN = CHWN with cuda-convnet
- BL_Best (highest bandwidth achieved in existing libraries) = cuDNN
  - While in fact according to (the experiment result of) the paper, it should be cuda-convnet
- Choose Layout = Choose implementations

Inconsistance



Found: 1

Done

Page 1

...ns on both single layers and various networks, with up to 27.9x for a single layer and up to 5.6x on the whole networks.

* Department of Integrated System, NEC

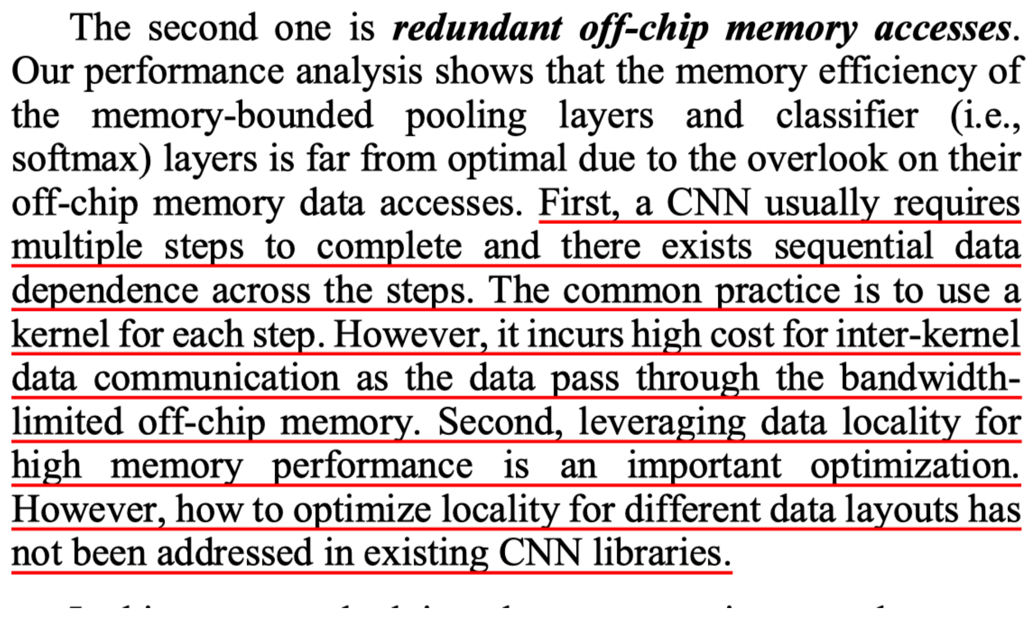
#{cli17, hzhou} @ ncsu.edu; *{yyang, mfeng,

***Abstract*— Leveraging large data sets, deep Convolutional Neural Networks (CNNs) achieve state-of-the-art recognition accuracy. Due to the substantial compute and memory operations, however, they require significant execution time. The massive parallel computing capability of GPUs make them as one of the ideal platforms to accelerate CNNs and a number of GPU-based CNN libraries have been developed. While existing works mainly focus on the computational efficiency of CNNs, the memory efficiency of CNNs have been largely overlooked. Yet CNNs have intricate data structures and their memory behavior can have significant impact on the performance. In this work, we study the memory efficiency of various CNN layers and reveal the performance implication from both data layouts and memory access patterns. Experiments show the universal effect of our proposed optimizations on both single layers and various networks, with up to 27.9x for a single layer and up to 5.6x on the**

studies o
complexit
coarse-gra
works mo
network,
efficiency
overlooke
structure,
not straigh
that have
memory e

Inconsistance

(Not sure) talk about something that is not directly relate to their work.



The second one is ***redundant off-chip memory accesses***. Our performance analysis shows that the memory efficiency of the memory-bounded pooling layers and classifier (i.e., softmax) layers is far from optimal due to the overlook on their off-chip memory data accesses. First, a CNN usually requires multiple steps to complete and there exists sequential data dependence across the steps. The common practice is to use a kernel for each step. However, it incurs high cost for inter-kernel data communication as the data pass through the bandwidth-limited off-chip memory. Second, leveraging data locality for high memory performance is an important optimization. However, how to optimize locality for different data layouts has not been addressed in existing CNN libraries.