High Performance Computing
9th Lecture

2016/10/28

YUKI ITO

Selected Paper:

vDNN: Virtualized Deep Neural Networks for Scalable, MemoryEfficient Neural Network Design

- Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, Stephen W. Keckler, NVIDIA
- Published as a conference paper at the 49th IEEE/ACM International Symposium on Microarchitecture (MICRO-49), 2016

Outline:

1. Introduction
2. Background and Motivation
3. Virtualized DNN
4. Methodology
5. Results
6. Related work
7. Conclusion

1. Introduction:

Deep neural networks (DNNs) have recently been successfully deployed in various application domains.

- Due to the tremendous compute horsepower offered by GPUs

Popular machine learning frameworks use GPUs for accelerated deep learning.

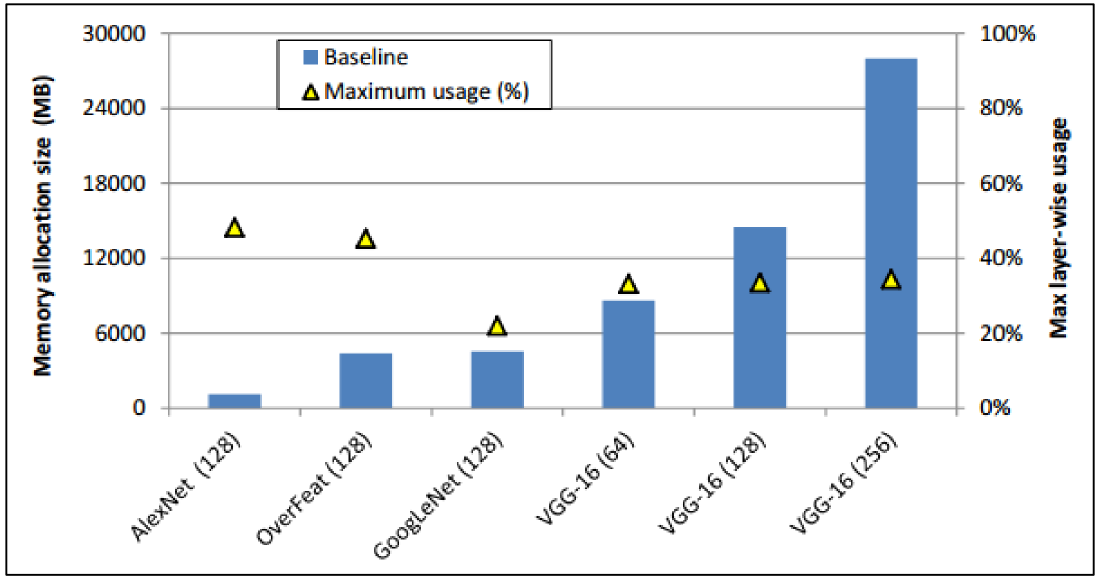
- The DRAM capacity of the GPUs limit the size of the DNN that can be trained.

1. Introduction:

e.g) Titan X: 12 GB memory capacity

The trend in deep learning is to move towards larger and deeper network designs.

- The physical memory limitations of GPUs is becoming important.



1. Introduction:

In this paper, authors propose ***vDNN***.

vDNN (virtualized Deep Neural Network)

- A runtime memory management solution that virtualizes the memory usage of DNN across both GPU and CPU memories.
- vDNN allows to train larger and deeper networks beyond the capacity of GPU.

2. Background and Motivation

DNNs are designed using a combination of multiple types of layers.

- Convolutional layer
- Activation layer
- Pooling layer
- Fully-connected layer.

feature extraction layers

classification layers

Convolutional neural networks are one of the most popular ML algorithms for image recognition.

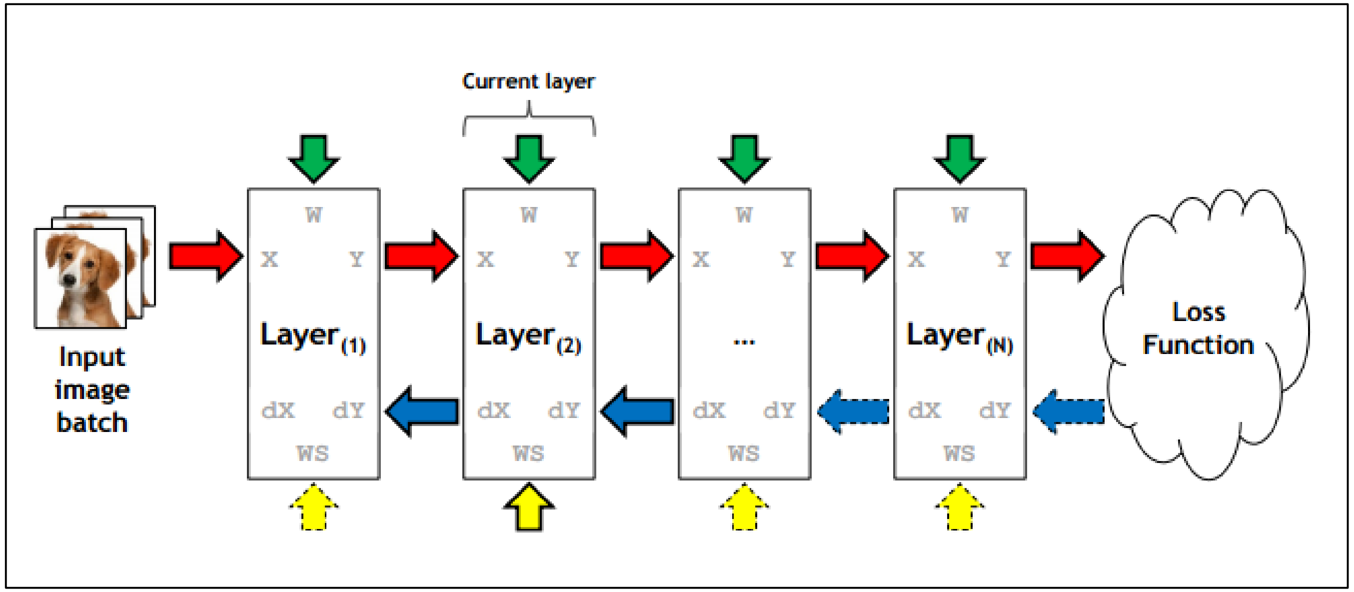
- These DNNs are trained using a backward propagation algorithm.

2. Background and Motivation

Forward propagation is performed from the first layer to the last layer.

Backward propagation is performed in the opposite direction.

- Both propagations traverse the network layer-wise.



2. Background and Motivation

Per layer memory allocations required are determined by the layer's input-output relationships and its mathematical function.

e.g) Convolutional layer

- Forward: input/output feature maps (X and Y), weights of the layer (W)
- Backward: input/output gradient maps (dY and dX), weight's gradient (dW), X and W
- If FFT based convolution algorithm is used, it needs an additional workspace (WS) buffer.

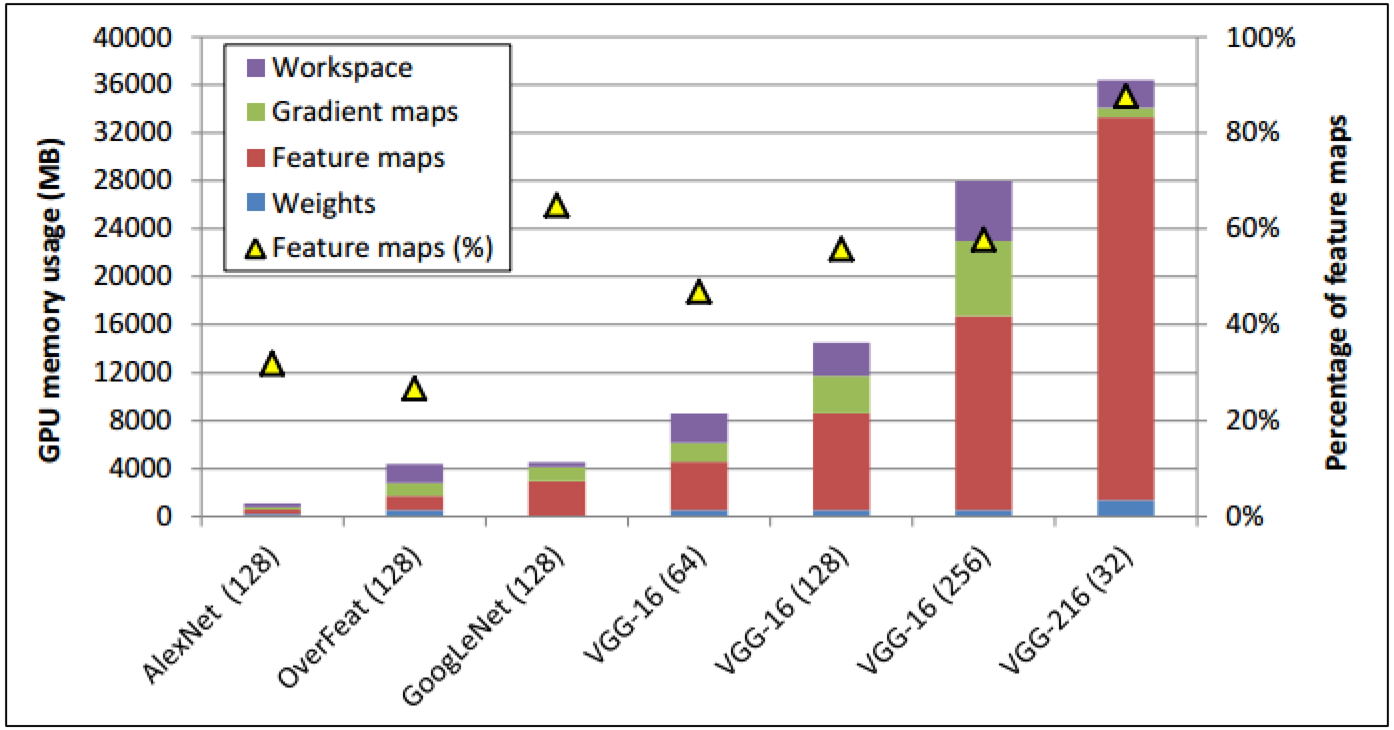
2. Background and Motivation

Because of the layer-wise gradient update rule of the backward propagation algorithm, each layer's feature maps (X) are later reused during its own backward propagation pass.

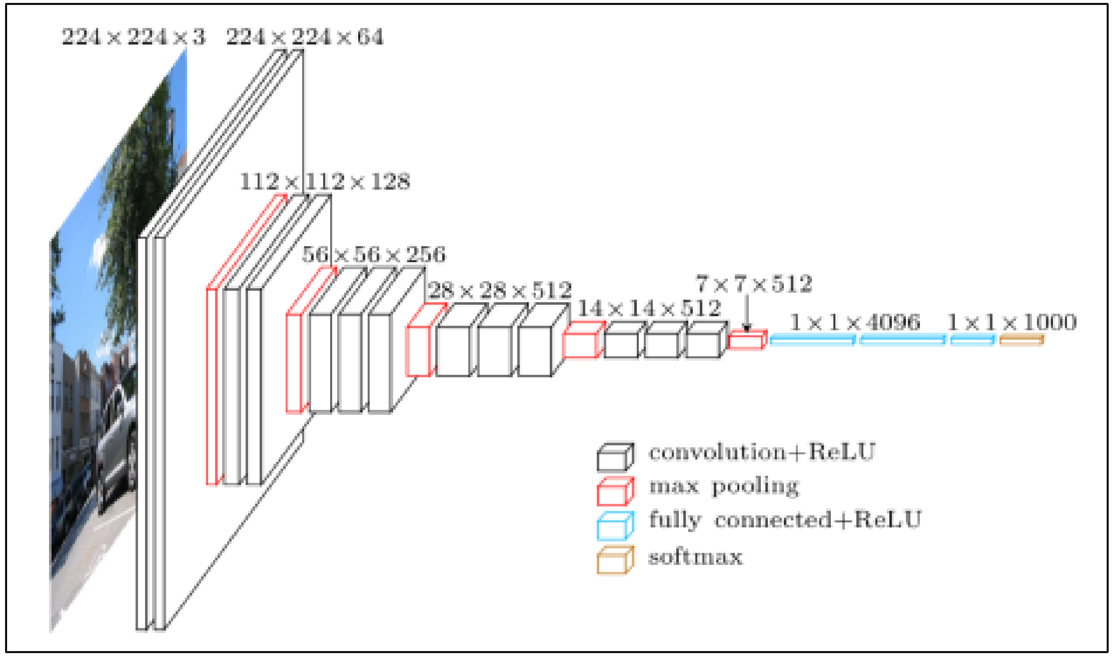
- All Xs must still be available in GPU memory until backward computation is completed.

As the number of layers increases, the fraction of memory allocated for feature maps grows.

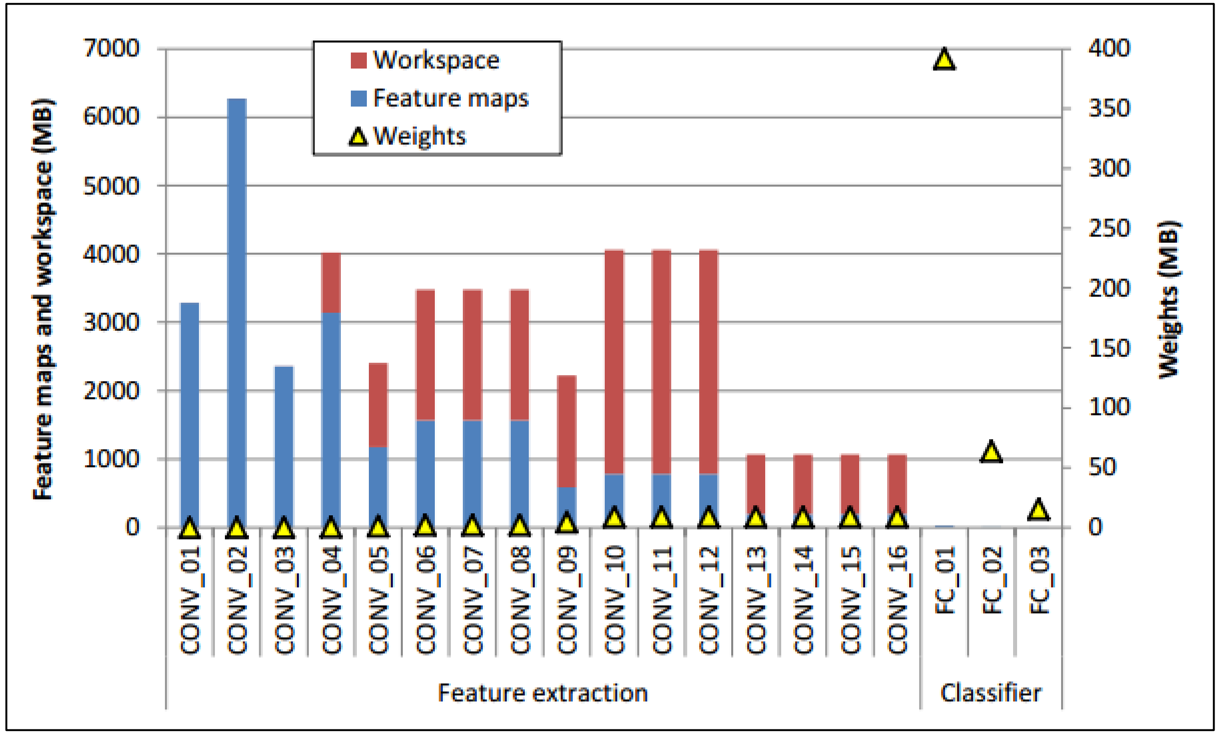
2. Background and Motivation



2. Background and Motivation



VGG network



Per layers memory usage of VGG-16
(batch size: 256) during forward

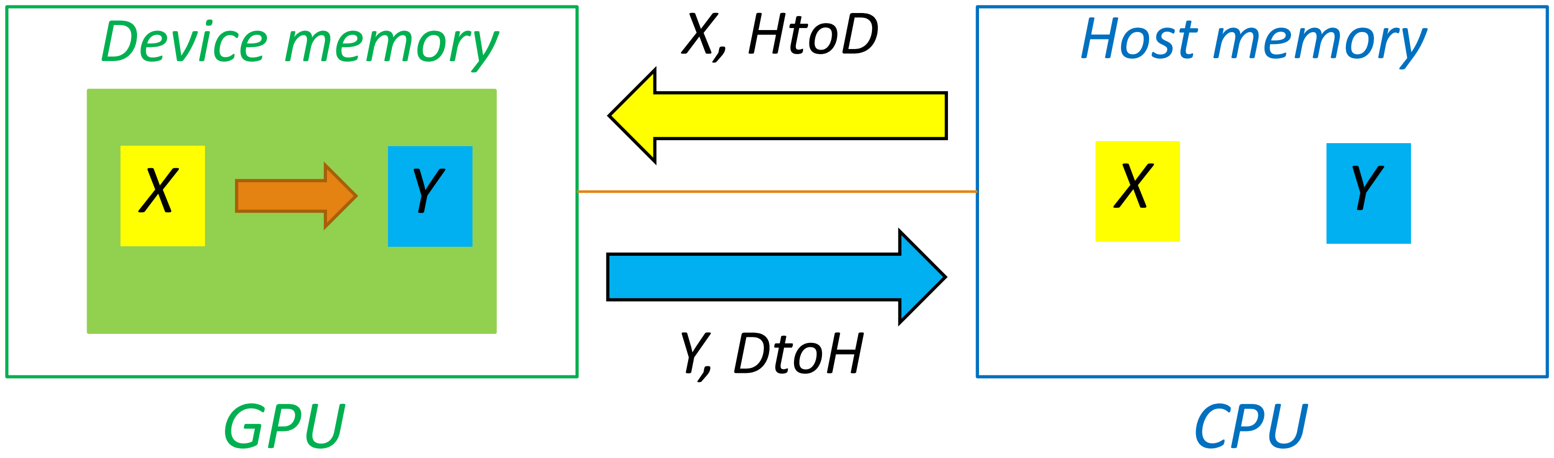
2. Background and Motivation

There are the following key observations about memory usage.

- The intermediate feature maps (X) and workspace (WS) incur higher memory usage compared to the weights (W) of each layer.
- Most of these X are concentrated on the feature extraction layers.
- Most of these W are concentrated on the later classifier layers.
- The per layer memory usage is much smaller than memory usage of the entire network.

3. virtualized DNN: *Design Principle*

The design objective of vDNN memory manager is to virtualize the memory usage of DNNs, using both GPU and CPU memory, while minimizing its impact on performance.



✓There is overhead due to communication between GPU and CPU.

3. virtualized DNN: *Design Principle*

vDNN is based on the three key observations.

1. DNNs are via SGD are designed and structured with multiple layers.
2. The training of these neural networks involves a series of layer-wise computations.
3. The GPU only processes a single layer's computation at any given time.

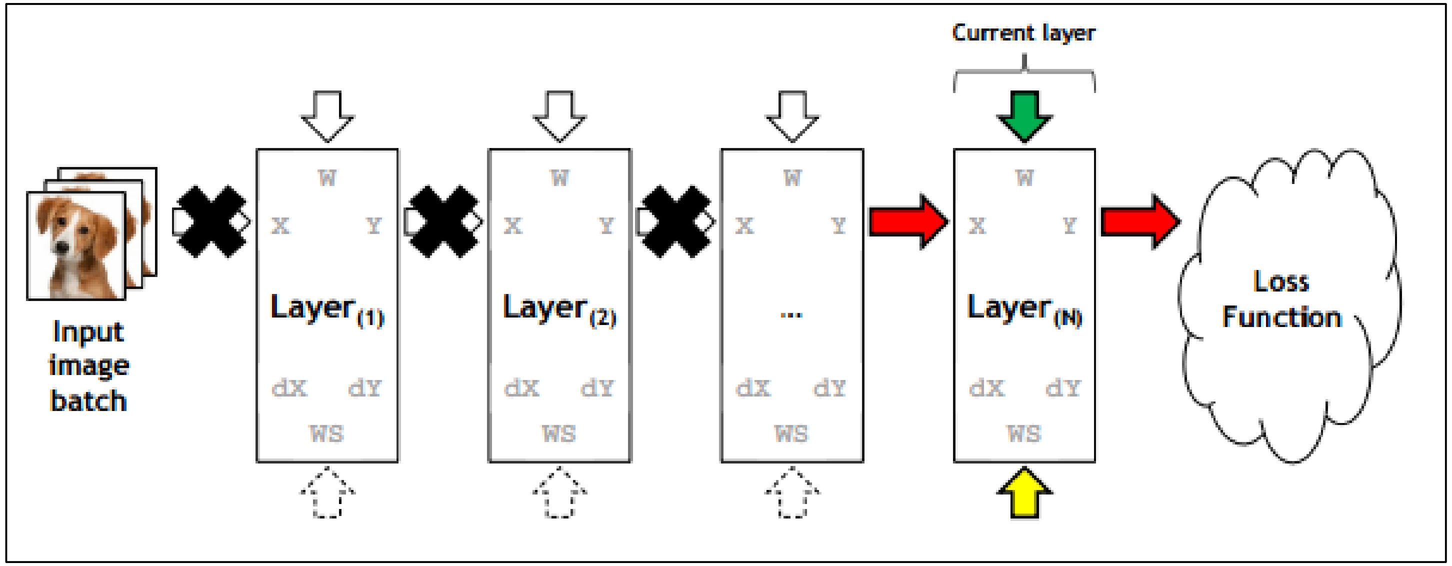
vDNN adopts a sliding-window based, layer-wise memory management strategy.

- The runtime memory manager allocates memory for the immediate usage of the layer that is currently being processed by the GPU.

3. virtualized DNN: *Design Principle*

Forward propagation:

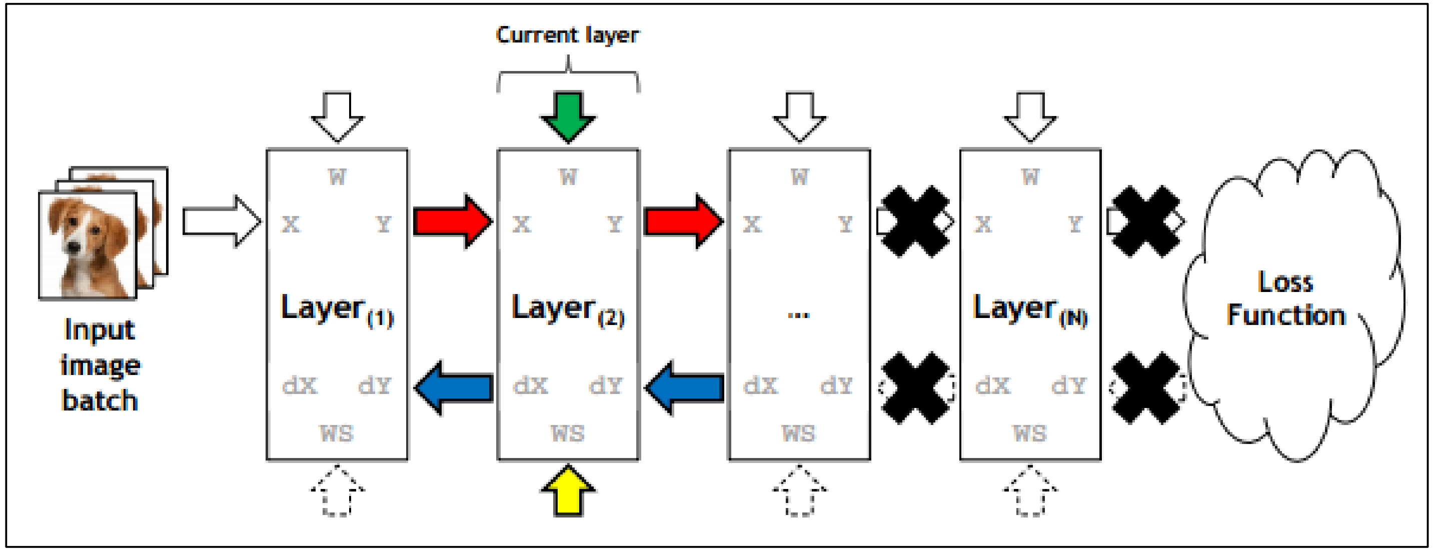
- vDNN allocates current layer's X on GPU.
- Other layer's Xs are offloaded to CPU memory.



3. virtualized DNN: *Design Principle*

Backward propagation:

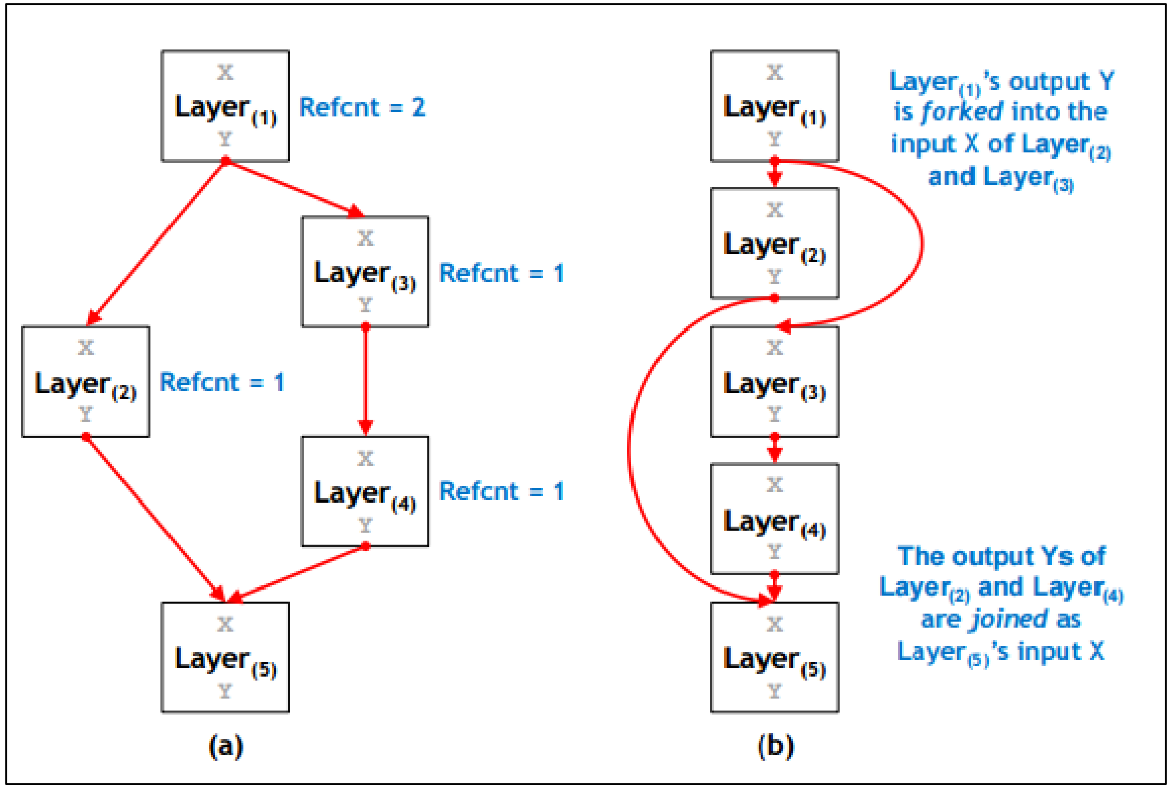
- Similar to forward propagation, vDNN aggressively releases data that are not needed for current layer's backward computation.



3. virtualized DNN: *Design Principle*

Non-linear feedforward network still involves a series of layer-wise computations.

- vDNN can also handle non-linear.



3. virtualized DNN: *Core Operations And Its Design*

vDNN uses cuDNN (https://developer.nvidia.com/cudnn) for computation on GPU.

- cuDNN is a GPU-accelerated library for DNN.
  - Various frameworks (including Caffe, Tensorflow, chainer) use cuDNN.

- cuDNN provide some algorithms for each layer's operation, and can find the best suited algorithm.

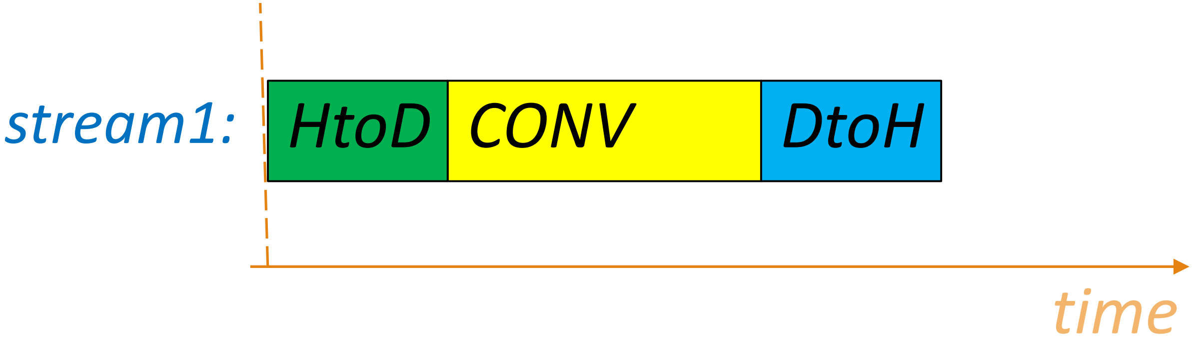
  e.g) convolutionForward: IMPLICIT_GEMM, GEMM, FFT, etc

3. virtualized DNN: *Core Operations And Its Design*

vDNN uses CUDA streams (https://docs.nvidia.com/cuda/cuda-c-programming-guide/).

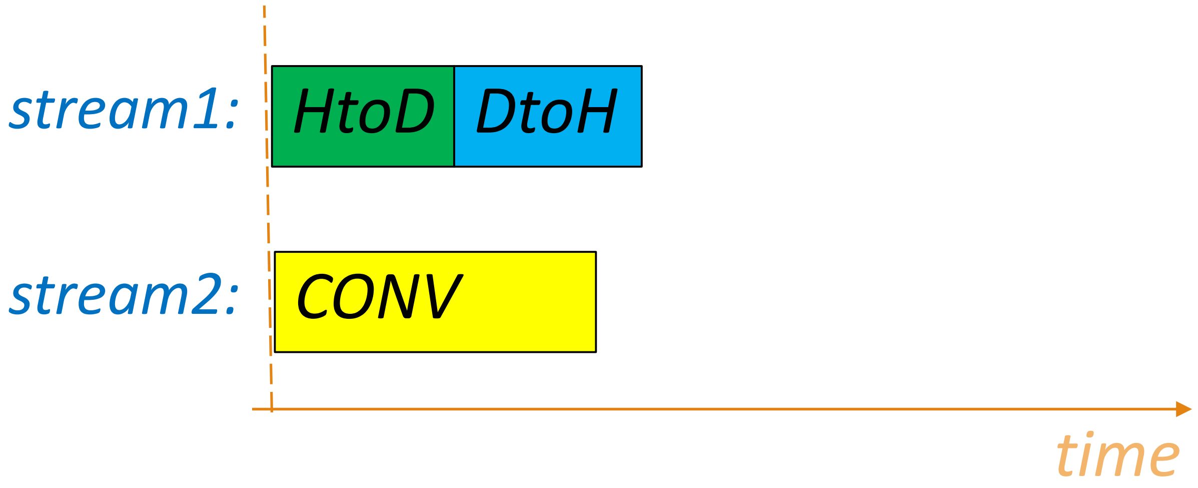
- A stream is a sequence of operations that execute in order on GPU.
- Different streams may execute their operations out of order with respect to one another or concurrently.

```
cudaMemcpyAsync(HtoD, stream1);

Convolution(stream1);

cudaMemcpyAsync(DtoH, stream1);
```



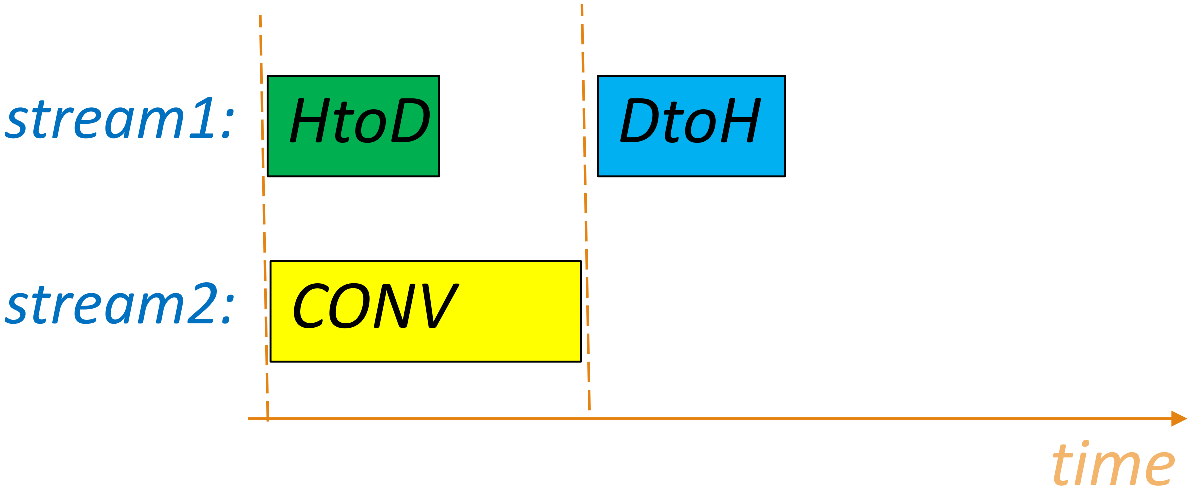
3. virtualized DNN: *Core Operations And Its Design*

```
cudaMemcpyAsync(HtoD, stream1);
Convolution(stream2);
cudaMemcpyAsync(DtoH, stream1);
```



3. virtualized DNN: *Core Operations And Its Design*

```
cudaMemcpyAsync(HtoD, stream1);
Convolution(stream2);
cudaStreamSynchronize(stream2);
cudaMemcpyAsync(DtoH, stream1);
```



3. virtualized DNN: *Core Operations And Its Design*

vDNN employs two streams, $stream_{compute}$ and $stream_{memory}$.

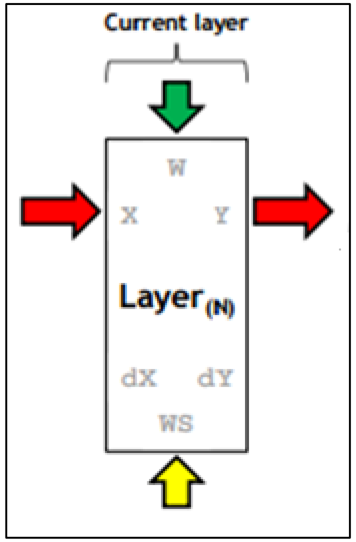
- $stream_{compute}$: all the layer's forward and backward computation
- $stream_{memory}$: the memory allocation/release, offload, and prefetch

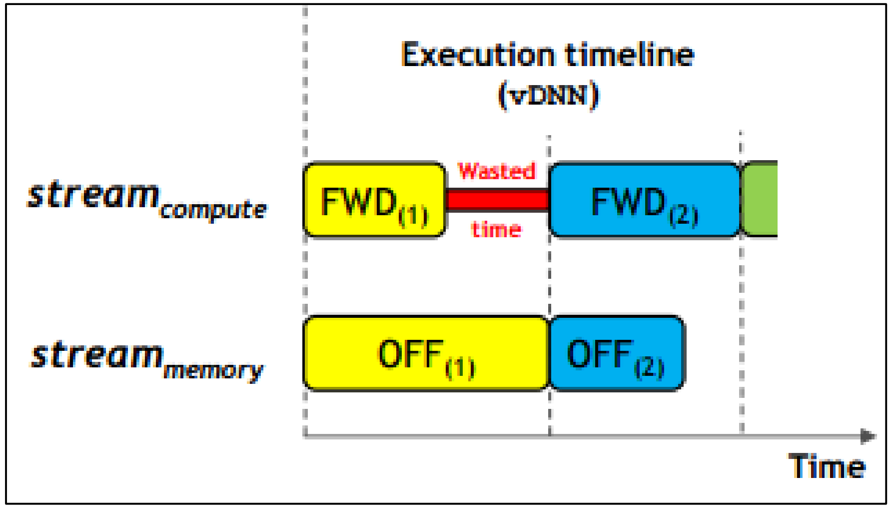
- Memory Allocation/Release
  - When the program launches, the vDNN allocates memory pool.
  - Whenever vDNN allocates (and releases) data structure, the memory is allocated (released) from memory pool without cudaMalloc() and cudaFree().

3. virtualized DNN: *Core Operations And Its Design*

Memory Offload

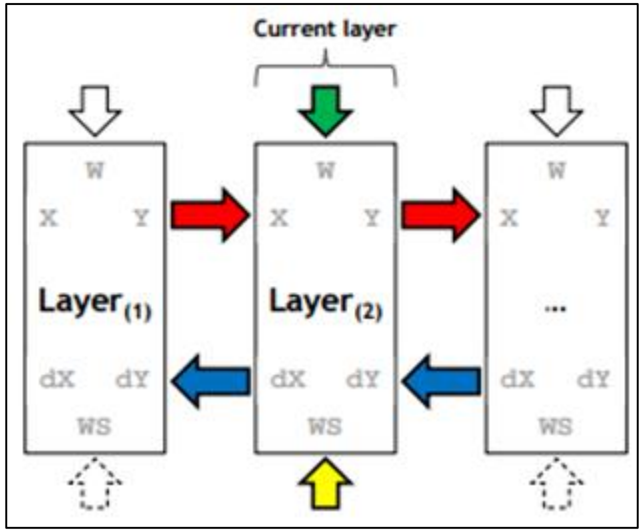
- Input feature maps (Xs) are offloaded from GPU to CPU.
- vDNN overlaps offloading with the same layer's forward computation.

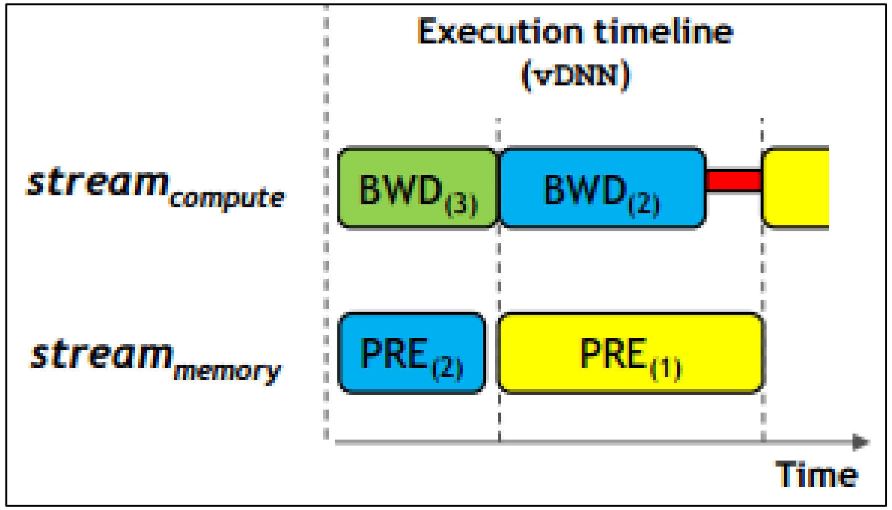




3. virtualized DNN: *Core Operations And Its Design*

- Memory Prefetch
  - Offloaded Xs are prefetched back from CPU to GPU.
  - vDNN overlaps other layer's prefetching with the current layer's backward computation.





3. virtualized DNN: vDNN Memory Transfer Policy

Determining the best layers to offload their X is a multi-dimensional optimization problem that must consider.

1. GPU memory capacity
2. The convolutional algorithms used and the overall layer-wise memory usage
   - “memory-optimal implicit GEMM” VS “performance-optimal convolutional algorithm”
3. The network-wide performance
   - The additional latency possibly incurred due to offload/prefetch

3. virtualized DNN: vDNN Memory Transfer Policy

Static vDNN

- *$vDNN_{all}$*
  - offload all layers' X from GPU
  - most memory-efficient solution

- *$vDNN_{conv}$*
  - only offload CONV layers' X from GPU
  - This policy is based on the observation that CONV layers have long computation latency to hide the latency of offload/prefetch.

3. virtualized DNN: vDNN Memory Transfer Policy

Static vDNN

- Convolutional algorithm is determined with memory-optimal or performance-optimal.

- While static vDNN is simple and easy to implement, it does not account for the system architectural components that determine the trainability and performance of a DNN.

3. virtualized DNN: vDNN Memory Transfer Policy

Dynamic vDNN

- *$vDNN_{dyn}$*
  - automatically determine the offloading layers and the convolutional algorithms at runtime
  - balance the trainability and performance of a DNN

- The dynamic vDNN launches profiling to optimization before training iterations.
  - This profiling is based on a greedy algorithm that tries to locally optimize layer’s memory usage and performance, seeking a global optimum state in terms of trainability and performance.

4. Methodology: GPU Node Topology

NVIDIA’s Titan X

- single precision throughput: 7 TFLOPS
- memory bandwidth: 336 GB/sec
- memory capacity: 12 GB

The GPU communicates with an Intel i7- 5930K via a PCIe switch.

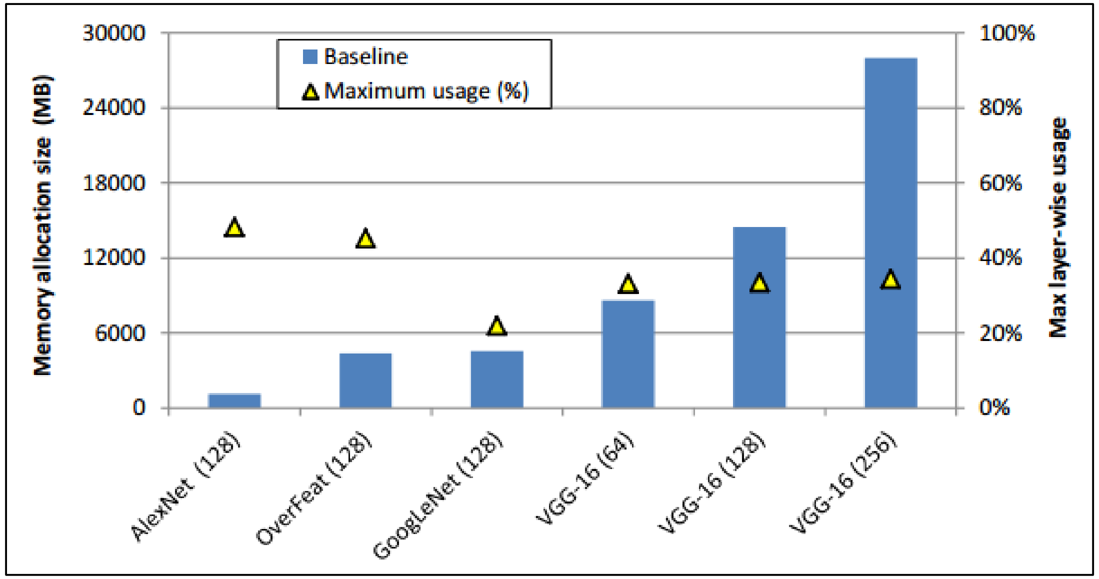
- bandwidth of communication with CPU: 16 GB/sec

4. Methodology: DNN Benchmarks

Conventional DNNs

AlexNet(128), OverFeat(128), Googlenet(128),

VGG-16(64), VGG-16(128), VGG-16(256)



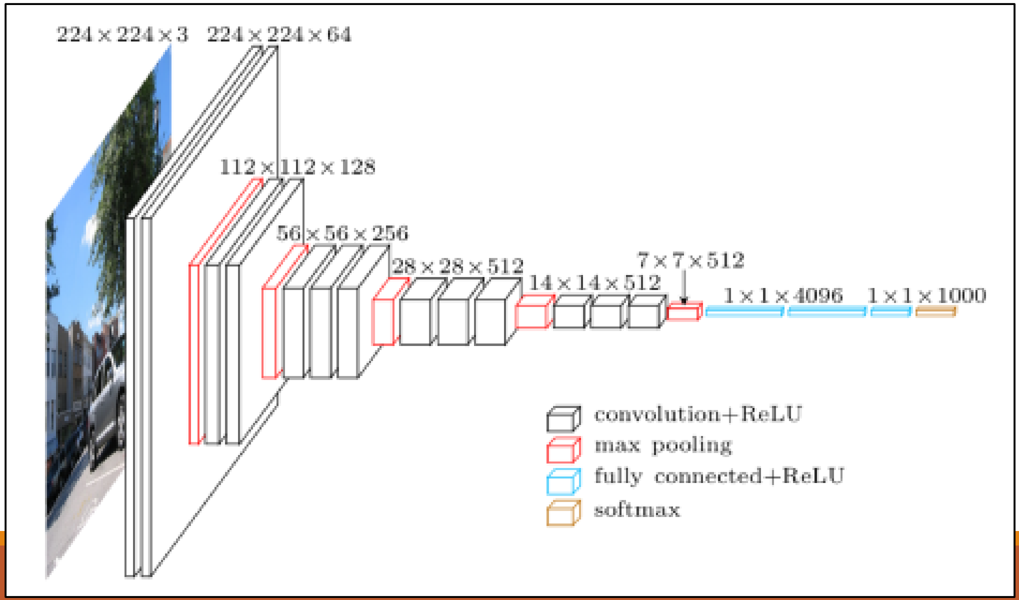
4. Methodology: DNN Benchmarks

Very Deep Networks

- extend the number of CONV layers of VGG

  VGG-116, VGG-216, VGG-316, VGG-416

- batch size: 32



5. Result

all: static $vDNN_{all}$

conv: static $vDNN_{conv}$

dyn: dynamic $vDNN_{dyn}$

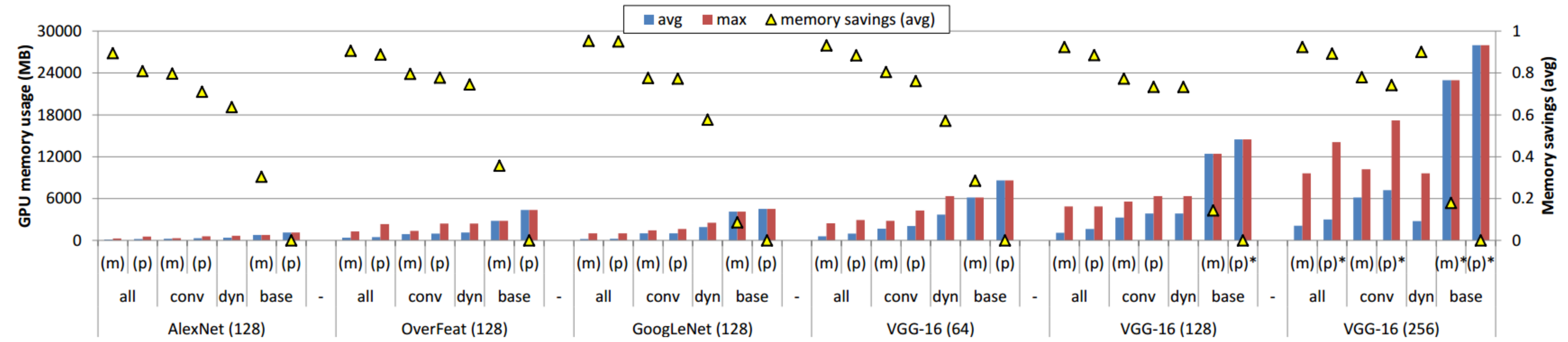
base: not use vDNN (All memory are allocated on GPU.)

- $vDNN_{all}$, $vDNN_{conv}$ and base are evaluated with both memory-optimal (m) and performance-optimal (p).

5. Result: GPU Memory Usage

Performance-optimal vDNN tend to allocate more memory on GPU to improve performance.

- Performance-efficient algorithms requires larger workspace.
- The total number of offload layers is reduced.

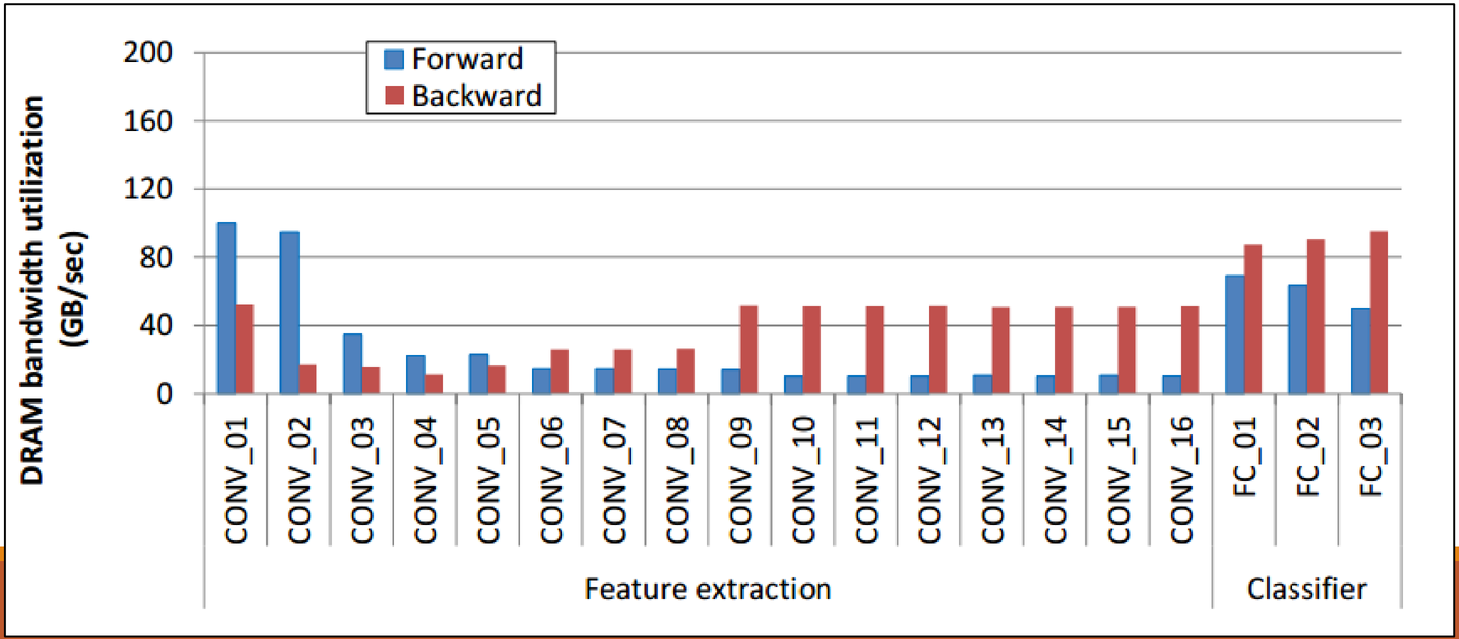


Average and maximum memory usage (left axis). Right axis corresponds to the savings in average memory usage.

5. Result: Impact on Memory System

vDNN does come at the cost of adding read/write traffic to the GPU memory subsystem.

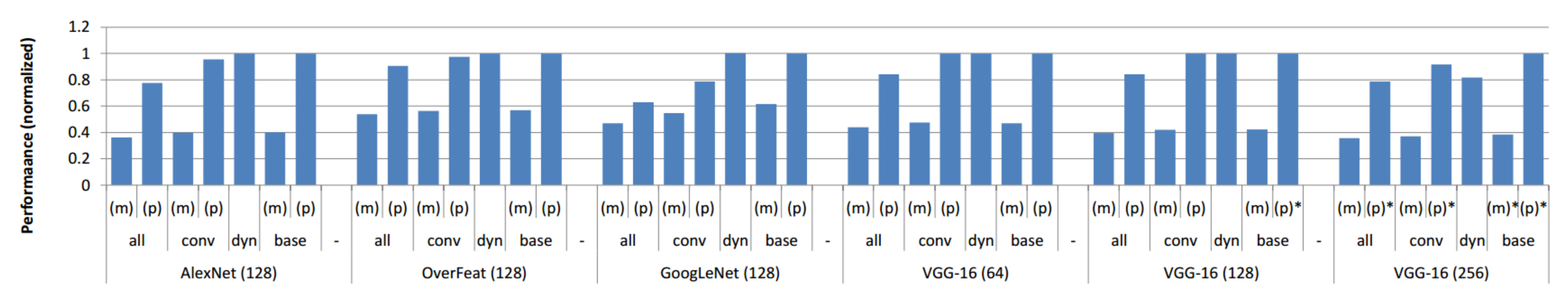
- Potentially interfering with the normal cuDNN operations.
- The feature extraction layers rarely saturate the 336 GB/sec of peak memory bandwidth.



5. Result: Performance

The *vDNN*$_{conv}$'s throughput reach an average 97 % of baseline's throughput.

The dynamic vDNN does much better in terms of balancing memory efficiency and overall throughput.



Overall performance (normalized to baseline).

5. Result: Power

The system profiling utility of *nvprof* is used to measure the average GPU power consumption (energy / time).

The additional energy overheads of $vDNN_{dyn}$ memory traffic is negligible on average.

- $vDNN_{dyn}$ dose not incur any noticeable performance overhead.
- The studied DNNs rarely saturate the peak DRAM bandwidth.

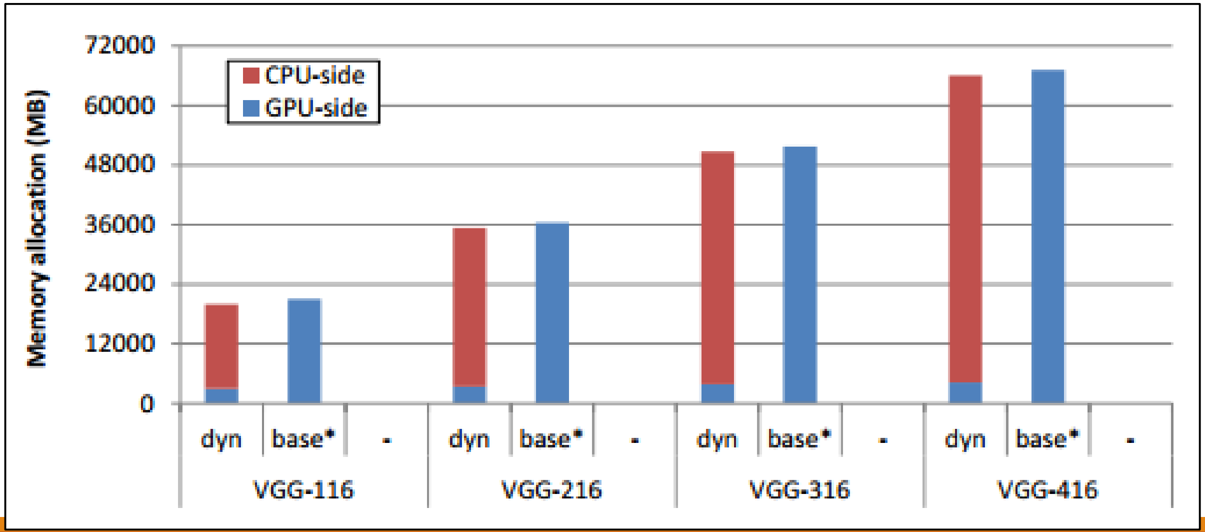
5. Result: Training Very Deep Networks

vDNN allocates most of memory in CPU memory.

- Very deep networks can be trained.

$vDNN_{dyn}$ did not incur any noticeable performance degradations.

- Because the offload/prefetch latency is completely hidden.



6. Related work

There have been a variety of proposals aiming to reduce the memory usage of neural networks.

- Network pruning techniques remove small valued weight connections.
- Reducing the number of bits required to model the network.

Several prior works discussed mechanisms to support virtualized memory on GPUs.

- TLB implementations that consider the unique memory access patterns of GPUs are proposed.

7. Conclusion

Existing ML frameworks require users to carefully manage their GPU memory usage.

- vDNN solution improves the memory-efficiency of DNN.

We also study the scalability of vDNN to extremely deep network.

- vDNN can train networks with hundreds of layers without any performance loss.