Split-CNN: Splitting Window-based Operations in Convolutional Neural Networks for Memory System Optimization

Tian Jin, Seokin Hong

24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS), 2019

Presenter: Hyun-Tak Lim

http://esoc.hanyang.ac.kr/people/hyuntak_lim/index.html

May 26, 2020

Neural Acceleration Study Season #2

Contents of presentation

- **Introduction**
- **Key contribution**
  - ❖ **Split-CNN**
  - ❖ **Heterogeneous Memory Management System(HMMS)**
- **Evaluation**
- **Conclusion**

Introduction

- **Memory Capacity Constraint**
  - Memory Bound Layers
  - Larger Batch Size
  - Higher Model Complexity

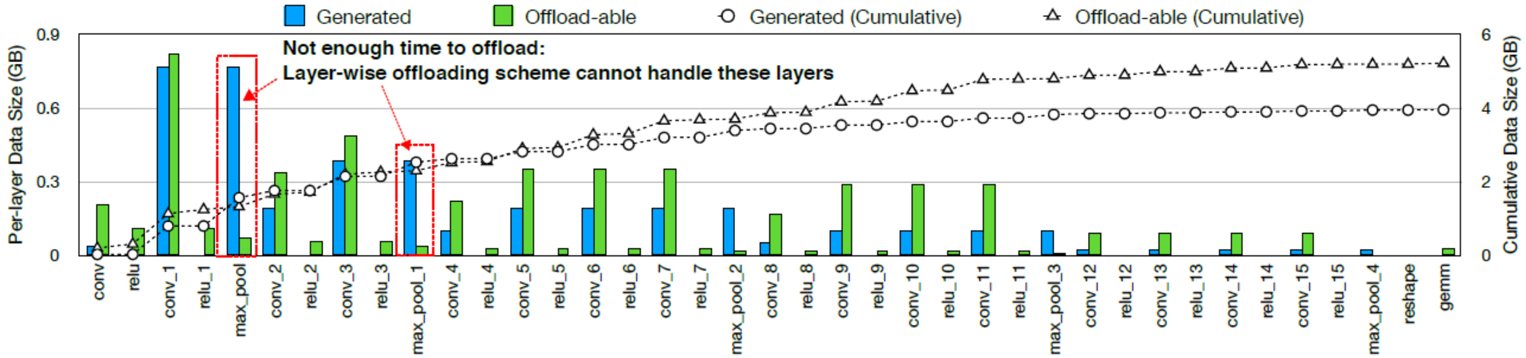
Introduction

- **Limitation of Layer-wise Allocation**

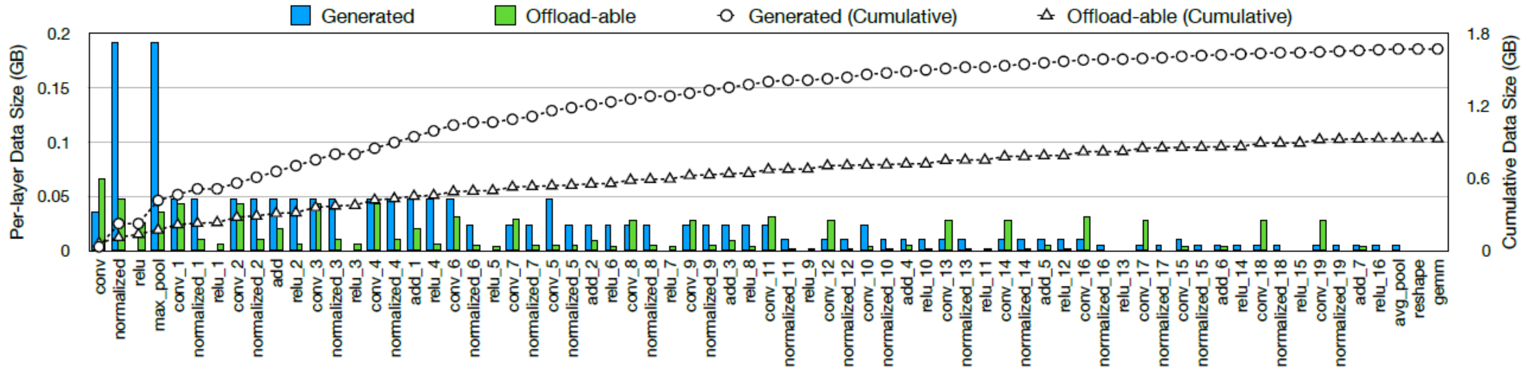
  - vDNN(sate of the art)
  - Less memory requirements
  - Some performance degradation
  - Require complex and multi-stage tuning process

Introduction

- **Opportunity**



(a) VGG-19 with batch-size=64



(b) ResNet-18 with batch-size=64

Split-CNN

- **Workflow**

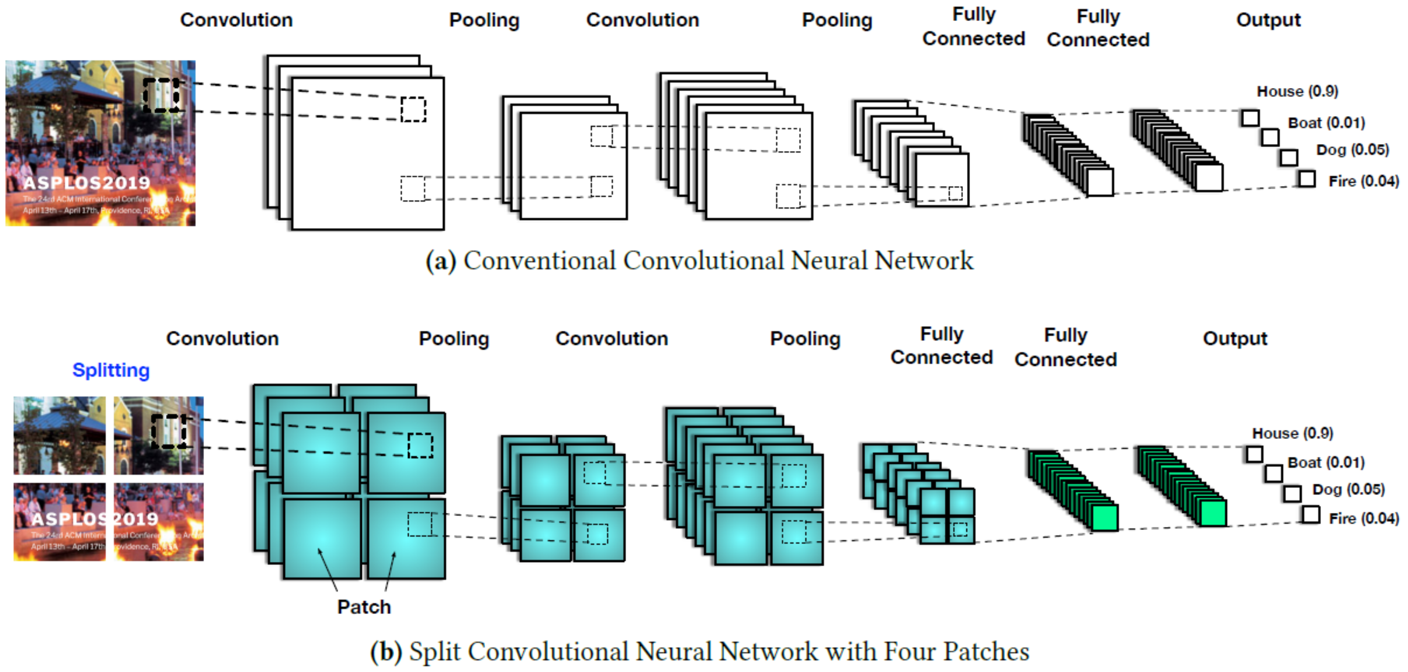


Figure 2. Conventional and Split Convolutional Neural Network

Split-CNN

- **Single Layer Formulation**

  - $Op(X, k, s, p)$
  - $Split_D\ (T, (s_0, \cdots, s_{N-1}))$
  - $[T_0, \cdots, T_n]_D$
  - $O = (O_0, \cdots, O_{N-1})$
  - $I = (I_0, \cdots, I_{N-1})$

Split-CNN

- **Single Layer Formulation**

$$lb\left(I_i\right) = O_i s - p_b \tag{1}$$

$$ub\left(I_i\right) = \left(O_i - 1\right) s + k - p_b \tag{2}$$

Split-CNN

- **Single Layer Formulation**

$$p_{i,b} = \begin{cases} p_b & i = 0 \\ I_i + p_b - (O_i - 1)\, s & otherwise \end{cases}$$

$$p_{i,e} = \begin{cases} p_e & i = N - 1 \\ (O_{i+1} - 1)\, s + k - (I_{i+1} + p_b) & otherwise \end{cases}$$

Split-CNN

- **Single Layer Formulation**

$$\boldsymbol{I} = ComputeInputSplitScheme\,(k, s, p, \boldsymbol{O}) \tag{3}$$

$$X_0, \cdots, X_{N-1} = Split_W\,(X, \boldsymbol{I}) \tag{4}$$

$$\boldsymbol{p} = (p_0, \cdots, p_{N-1}) = ComputePadding\,(k, s, p, \boldsymbol{O}, \boldsymbol{I}) \tag{5}$$

$$\forall n \in \{0, \cdots, N-1\}\; Y_n = Op\,(X_n, k, s, p_i) \tag{6}$$

$$Y = [Y_0, \cdots, Y_{N-1}]_W \tag{7}$$

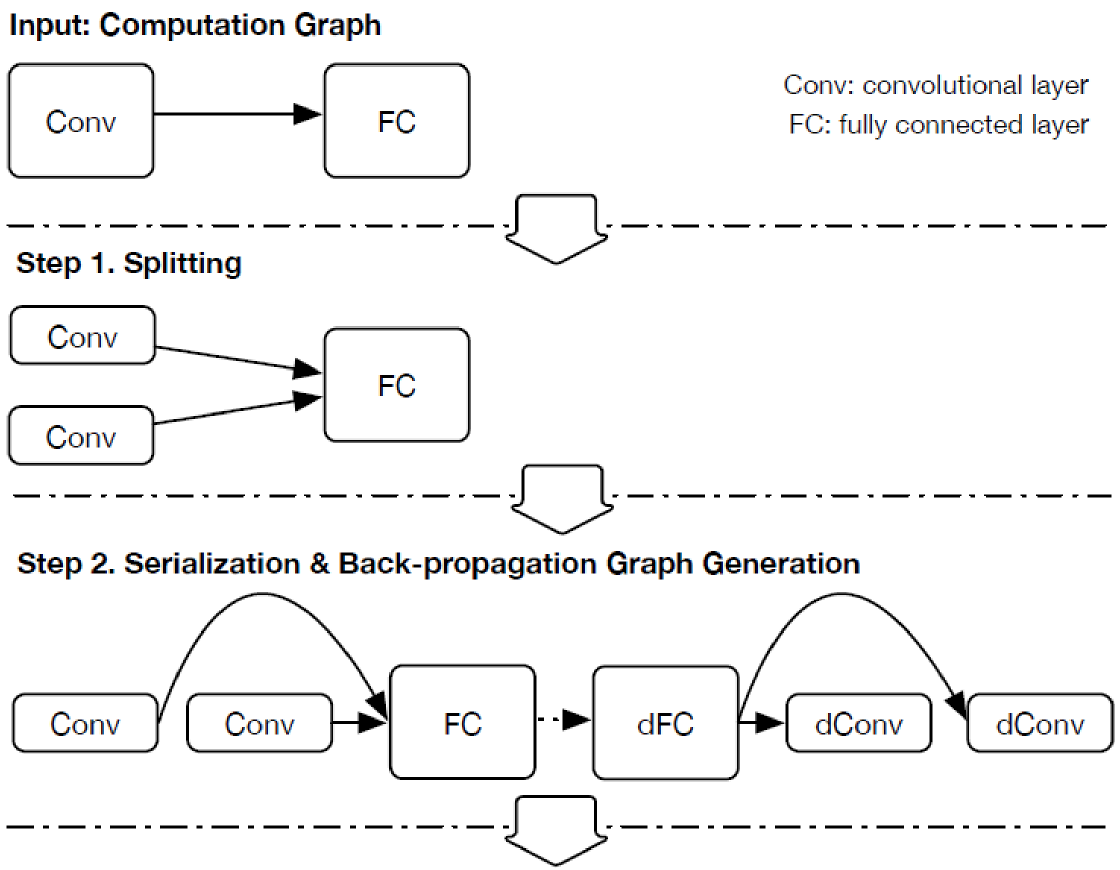
Split-CNN

- **Stochastic Splitting**

$$s_i \sim DiscreteUniform\left(\lceil\frac{(i-\omega)\cdot L}{N}\rceil, \lfloor\frac{(i+\omega)\cdot L}{N}\rfloor\right)$$

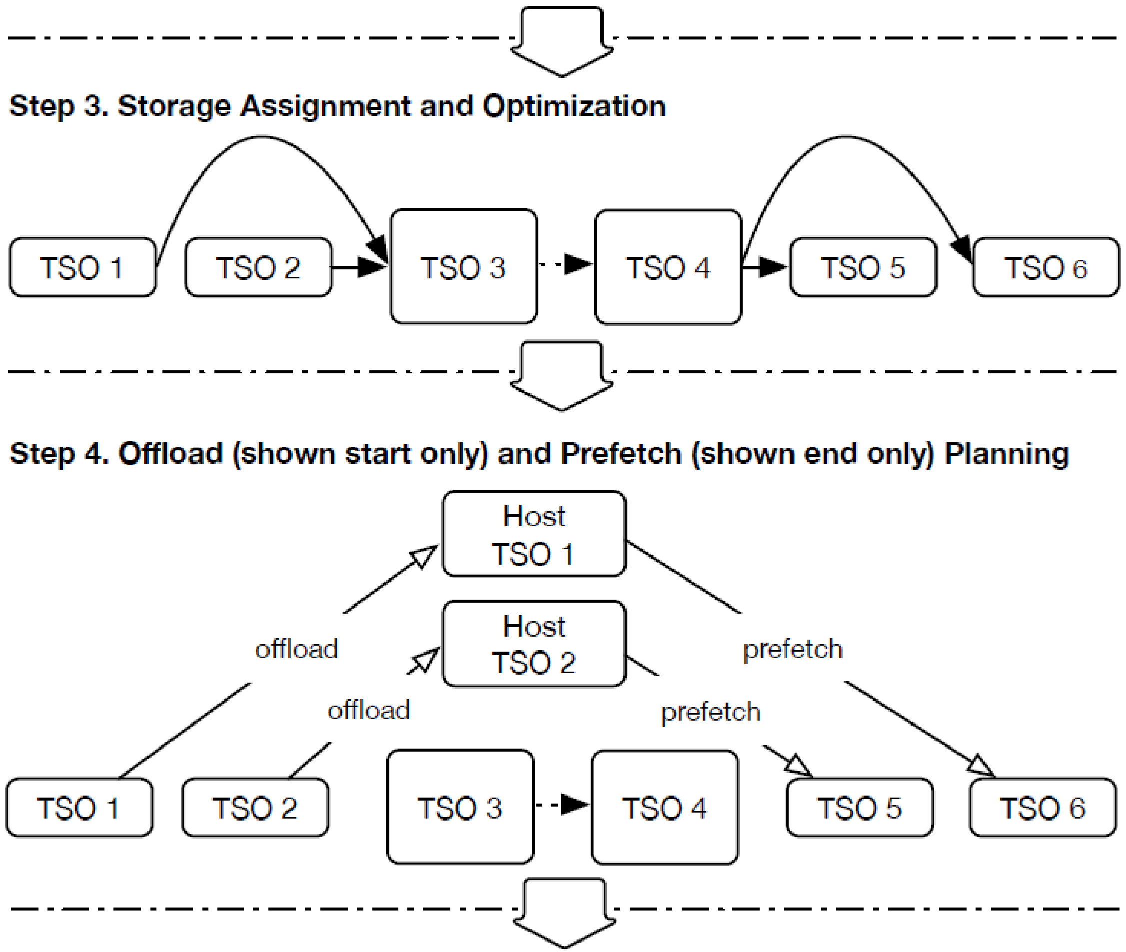
HMMS: Heterogeneous Memory Management System

- **Workflow**



HMMS: Heterogeneous Memory Management System

- **Workflow**



HMMS: Heterogeneous Memory Management System

- **Workflow**

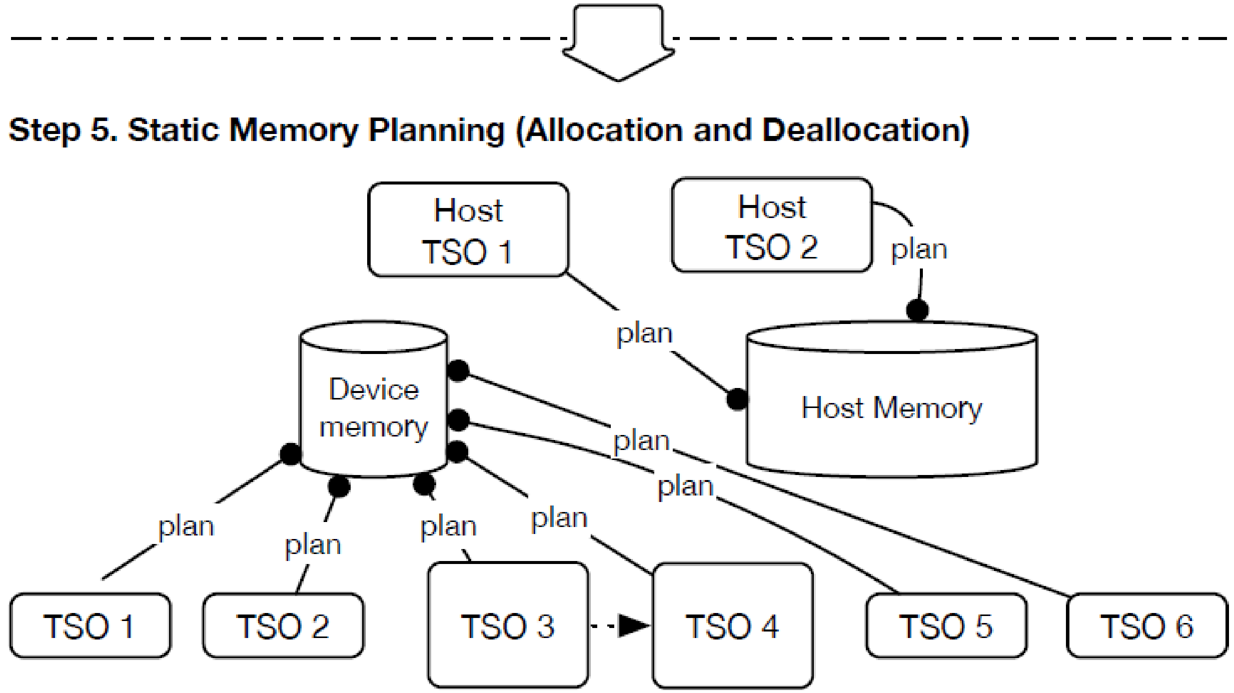


Figure 3. Heterogeneous Memory Management System (HMMS)

HMMS: Heterogeneous Memory Management System

- **Computation Graph**
  - G = (N,E)
  - Directed Acyclic Graph
- **Tensor Storage Object**
  - Separate conceptual and physical tensor
  - TSO represent a contiguous region of memory storage space

HMMS: Heterogeneous Memory Management System

- **Splitting and Graph Generation**
  - Split training model
  - HMMS automatically transforms regular CNN to Split-CNN
  - Serialize computation by topological sort

HMMS: Heterogeneous Memory Management System

- **Storage Assignment and Optimization**

  - Assign each tensor in graph to a TSO
  - Keep reference counter for each TSO
  - Memory Optimization
    1) **In-Place ReLu**
    2) **Summation Error Storage Object Sharing**

HMMS: Heterogeneous Memory Management System

- **Offload and Prefetch Planning**

  - *Start of Offload* : kick off the device to host memory transfer through idle memory stream
  - *End of Offload* : synchronize computation stream with memory stream through which TSO of interest is transferred
  - *Start of Prefetch* : retrieve content of TSO from host back to device via idle memory stream
  - *End of Prefetch* : synchronize compute stream and memory stream before TSO appear as storage object

HMMS: Heterogeneous Memory Management System

- **Offload and Prefetch Planning**

```
Algorithm 1: Offload Planning Algorithm
Data: serialized list of (fwd) operations in the CNN : ops
Initialize offload_capacity_balance = 0 ;
Initialize TSO_to_free = {} ;
Initialize profile_exec_time as described ;
Initialize nvlink_bandwidth as described ;
// Initialize memory and computation streams.
Initialize mem_stream[], comp_stream ;
for Operation op ∈ ops do
    input_TSO = TSO of input feature map of op ;
    if no further write happens to input_TSO then
        Get an idle memory stream m.
        Plan to allocate host TSO for input_TSO
          immediately before op starts executing. ;
        Plan to transfer input_TSO to host via m
          immediately after op starts executing. ;
        input_TSO.stream = m. ;
        Append input_TSO to TSO_to_free. ;
        offload_capacity_balance -= input_TSO.size ;
    end
    // Compute the increase of offloading capacity by
      multiplying op execution time with nvlink
      bandwidth.
    increase = profile_exec_time[op] *
      nvlink_bandwidth;
    offload_capacity_balance += increase ;
    if offload_capacity_balance ≥ 0 or op is the last then
        for TSO tso ∈ TSO_to_free do
            Plan to synchronize with tso.stream
              immediately after op starts executing. ;
            Plan to free tso immediately after the above
              synchronization. ;
        end
        if TSO_to_free is not empty then
            Plan to synchronize with comp_stream after
              above synchronizations with memory
              stream. ;
            offload_capacity_balance = 0 ;
            Clear TSO_to_free.
        end
    end
end
```

HMMS: Heterogeneous Memory Management System

- **Static Memory Planning**

  - Three memory pools

    1) Host general purpose memory
    2) Device parameter memory
    3) Device general purpose memory

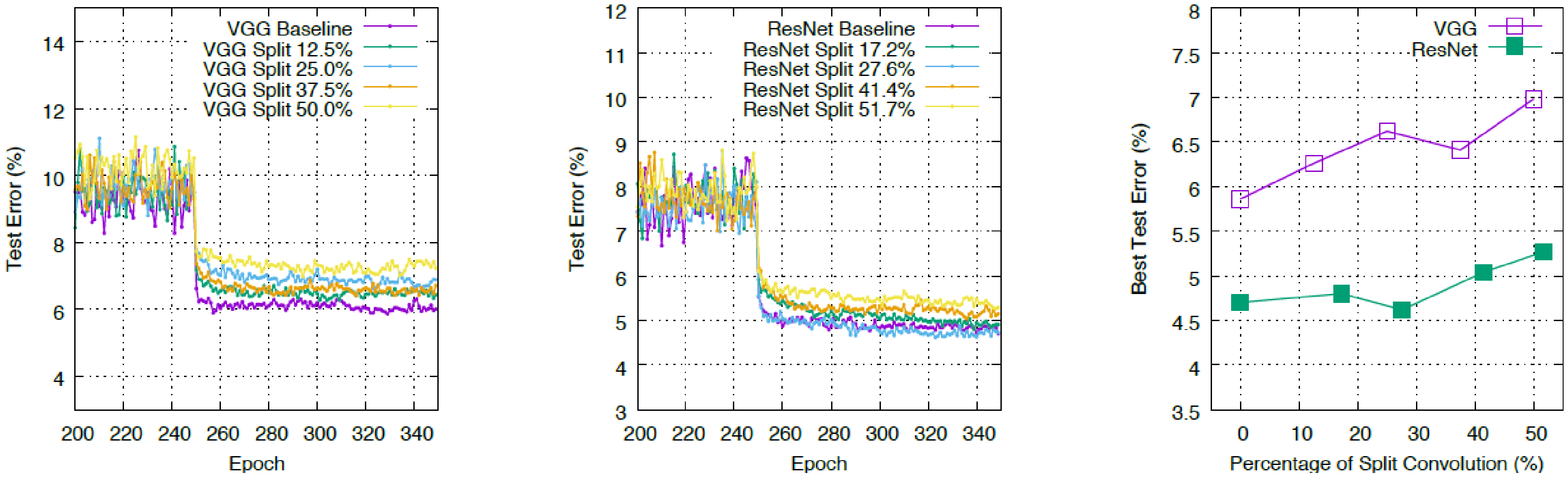
Evaluation: Split-CNN

- **Methodology**

  - Model : AlexNet, VGG-19, ResNet-18, ResNet-50
  - Batch size : 256 for ImageNet, 128 for CIFAR-10
  - Splitting depth : 0%, 12.5%, 25.0%, 37.5%, 50%
  - # of Splits : 1, 2, 3, 4, 6, 9

Evaluation: Split-CNN

- **Split Depth**

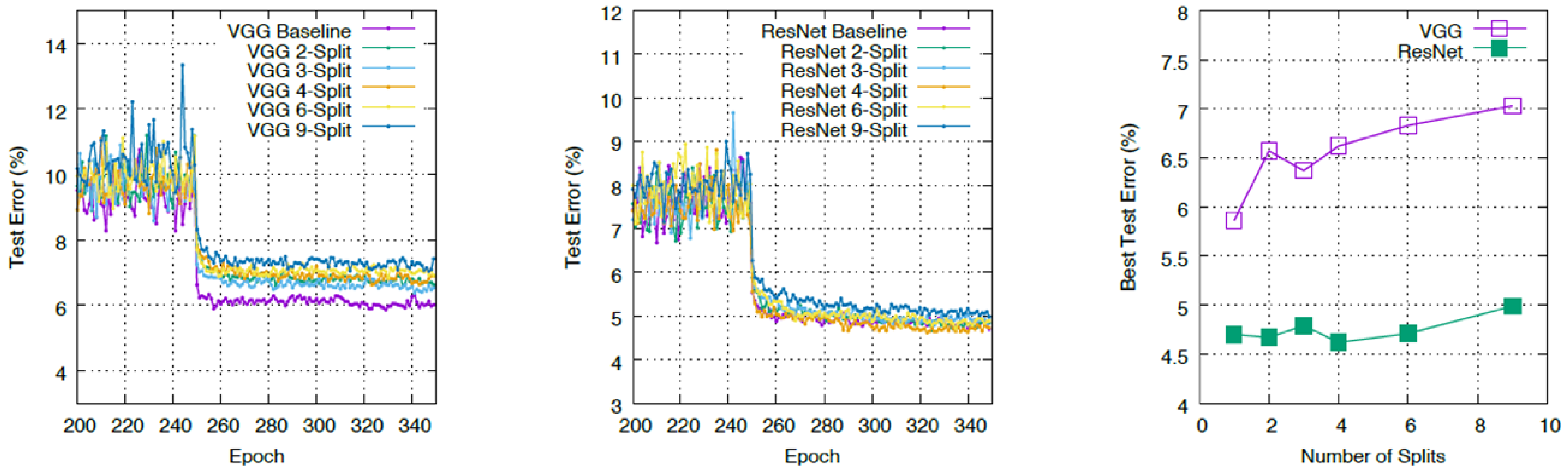


(a) Split-VGG-19 w/ Various Depths (b) Split-ResNet-18 w/ Various Depths (c) Best Test Error vs. % Split

Figure 4. Effects of Splitting Depth on Test Error (lower is better)

Evaluation: Split-CNN

- **Number of Split**

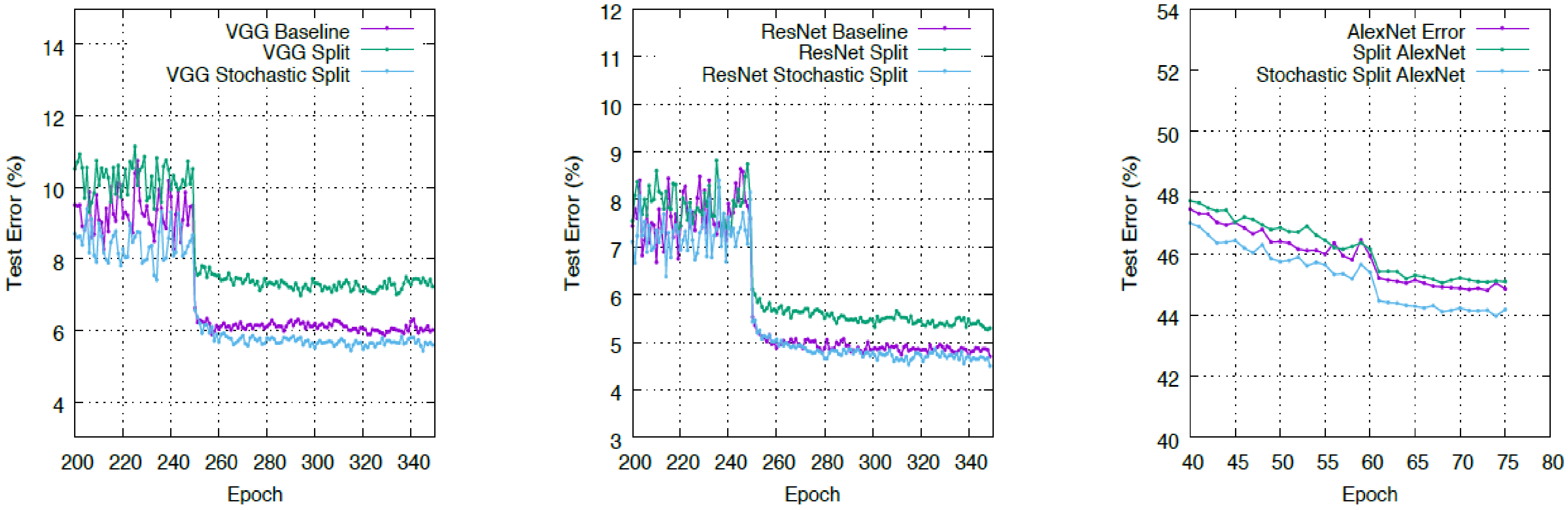


(a) Split-VGG-19 w/ Various No. Splits **(b)** Split-ResNet-18 w/ Various No. Splits **(c)** Best Test Error vs. % Split

Figure 5. Effects of Number of Splits on Test Error (lower is better)

Evaluation: Split-CNN

- **Stochasticity**

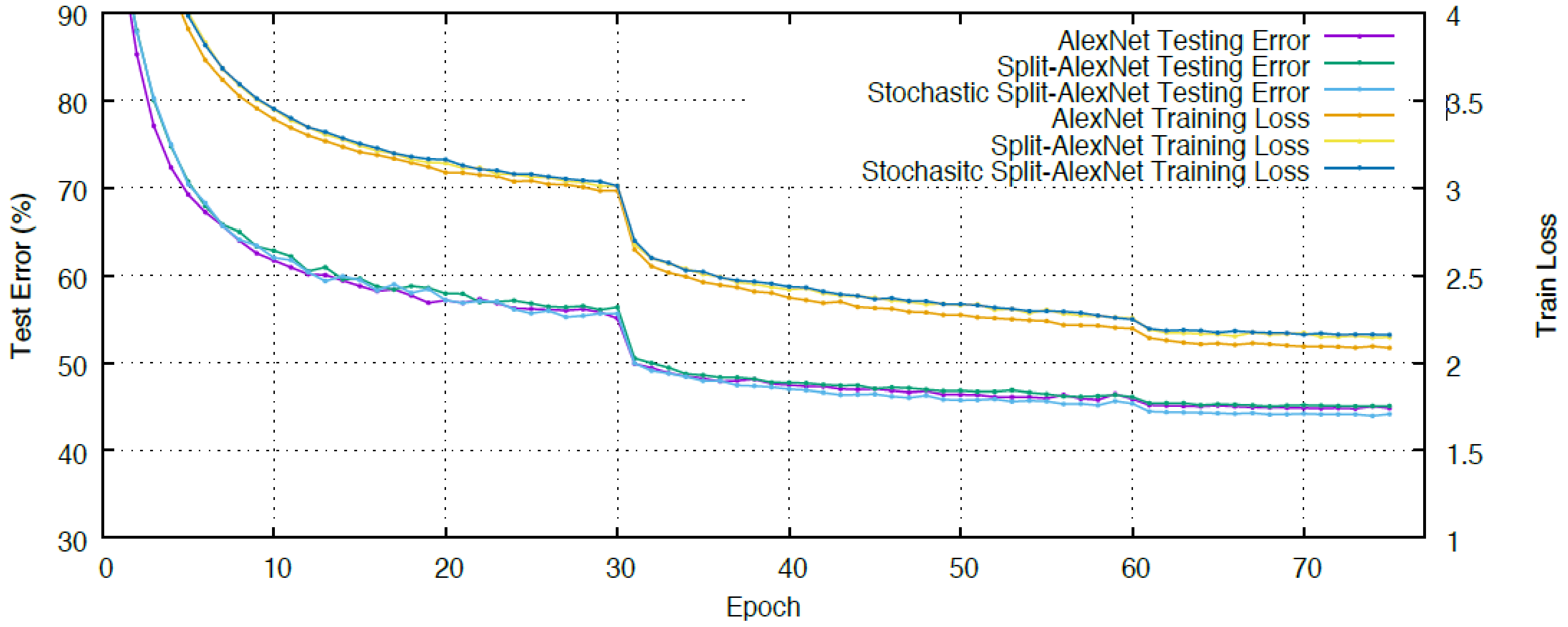


(a) VGG-19 (b) ResNet-18 (c) AlexNet

Figure 6. Effects of Stochasticity of Splitting on Test Error (lower is better)

Evaluation: Split-CNN

- **Performance**



Evaluation: Split-CNN

- **Performance**

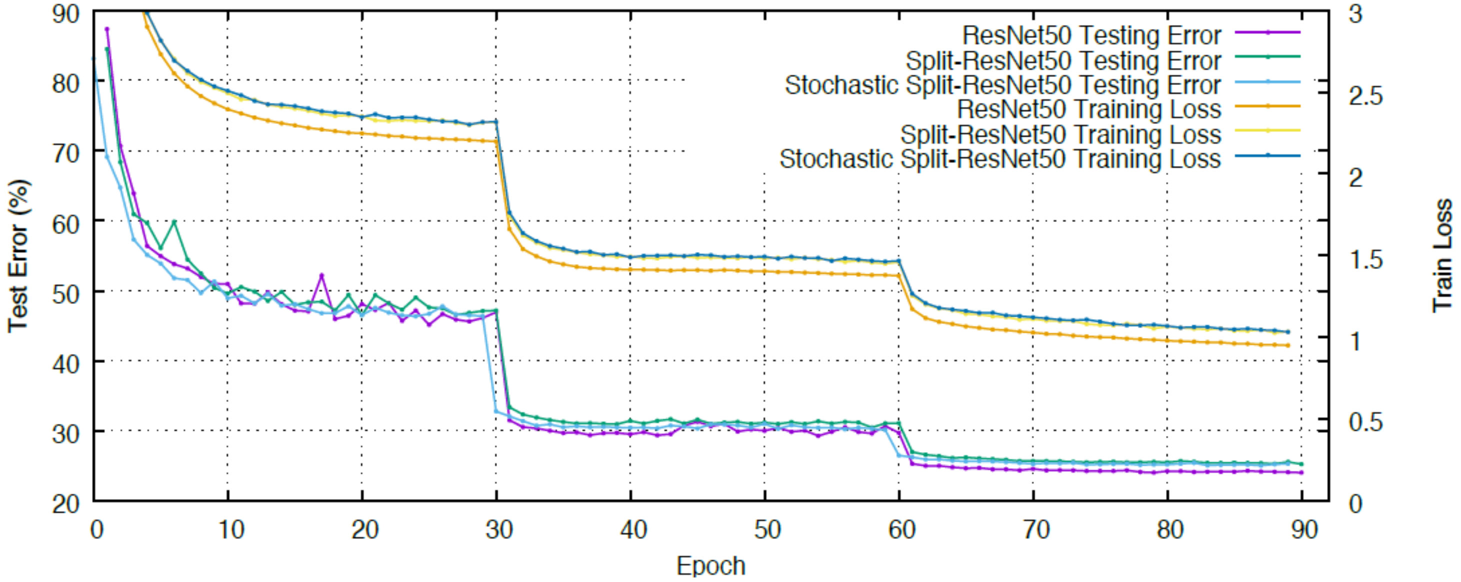


Figure 7. Split-CNN Classification Performance on ImageNet

Evaluation: Split-CNN

- **Performance**

Classification Accuracy of Split-CNN				
Architecture	AlexNet	ResNet50	VGG19	ResNet18
Dataset	ImageNet	ImageNet	CIFAR	CIFAR
Splitting Depth	60%	81.2%	50 %	50 %
No. of Splits	4	4	4	4
Baseline Acc.	55.2 %	**75.9 %**	94.14 %	95.3 %
SCNN Acc.	55.0 %	74.7 %	93.02 %	94.8 %
SSCNN Acc.	**55.9 %**	74.9 %	**94.58 %**	**95.5 %**

Table 1. Classification Performance of Split-CNN

Evaluation: HMMS

- **Methodology**

  - IBM Power System S822LC with T16GB esla P100 GPUs
  - IBM Power8 CPU
  - NVLink 1.0, bandwidth 34.1GB/s
  - NVIDIA cuDNN V7

Evaluation: HMMS

- **Training Throughput**

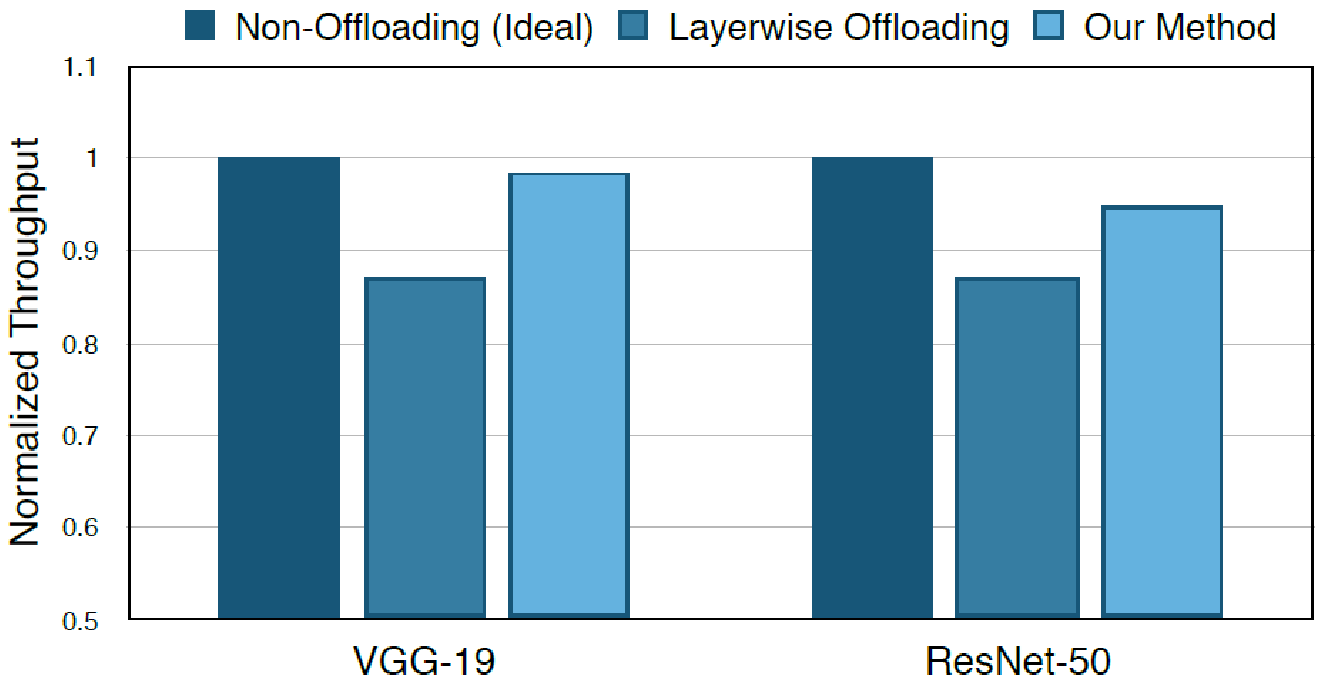


Figure 8. Training Throughput with Three Scheduling Methods

Evaluation: HMMS

- **Profiling Results**

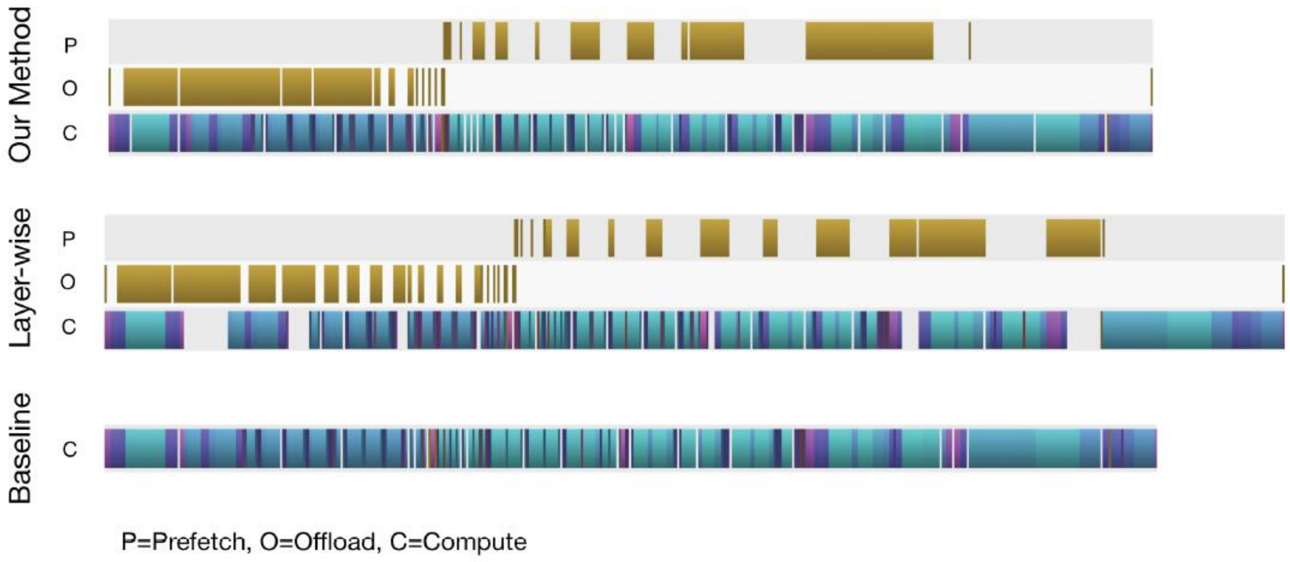


Figure 9. Profiling Results for VGG-19 with Three Offload-Scheduling Methods

Evaluation: HMMS

- **Maximum Batch**

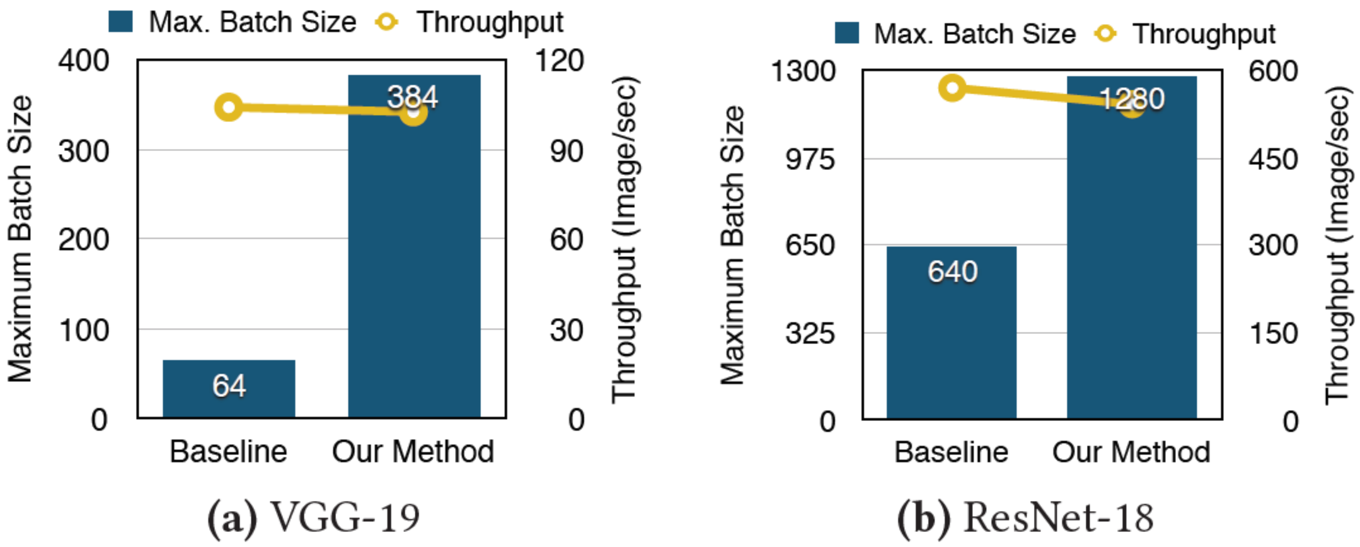


Figure 10. Impact on the Maximum Batch Size and Throughput with number of splits = 4, depth ≈ 75%

Evaluation: HMMS

- **Speedup**

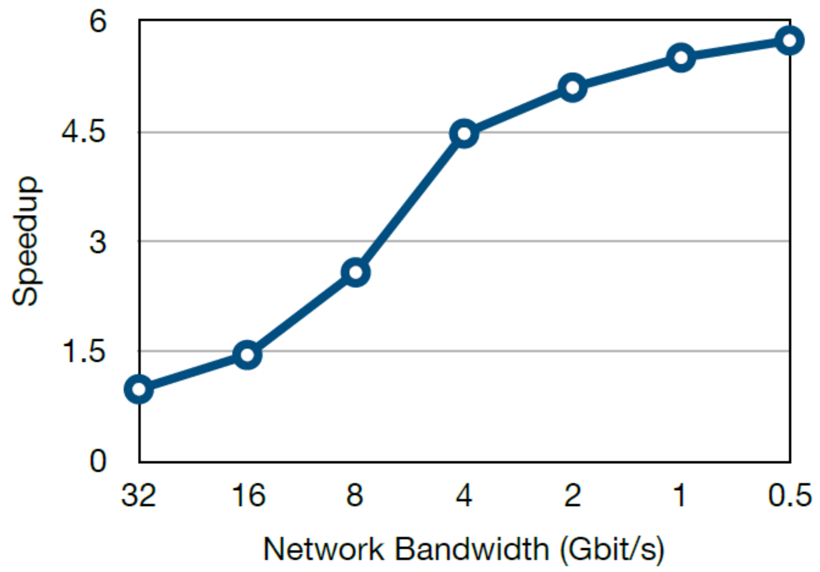


Figure 11. Speedup of Distributed Training with Split-CNN

Conclusion

- Presented Split-CNN
- Stochastic Split-CNN can enhance performance metrics
- Proposed HMMS
- Enabled training VGG-19 with 6x larger batch size, VGG-18 with 2x larger batch sizse

Thank you