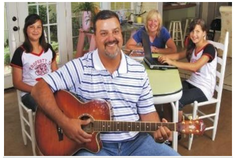
Lecture 12: Model Serving

CSE599W: Spring 2018

Deep Learning Applications

"That drink will get you to 2800 calories for today"

"I last saw your keys in the store room"

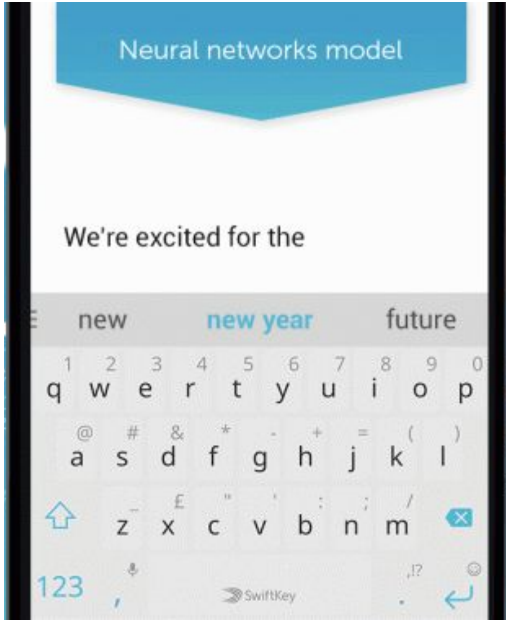


"Remind Tom of the party"

"You're on page 263 of this book"

Intelligent assistant

Surveillance / Remote assistance

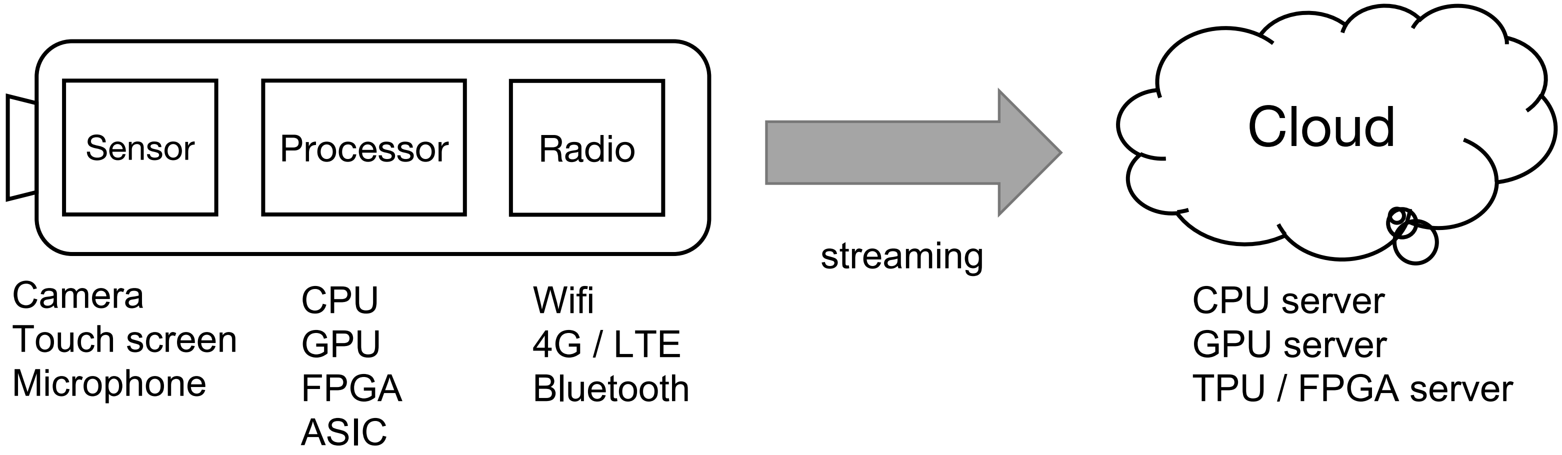


Input keyboard

Model Serving Constraints

- Latency constraint
  - Batch size cannot be as large as possible when executing in the cloud
  - Can only run lightweight model in the device
- Resource constraint
  - Battery limit for the device
  - Memory limit for the device
  - Cost limit for using cloud
- Accuracy constraint
  - Some loss is acceptable by using approximate models
  - Multi-level QoS

Runtime Environment



Resource usage for a continuous vision app

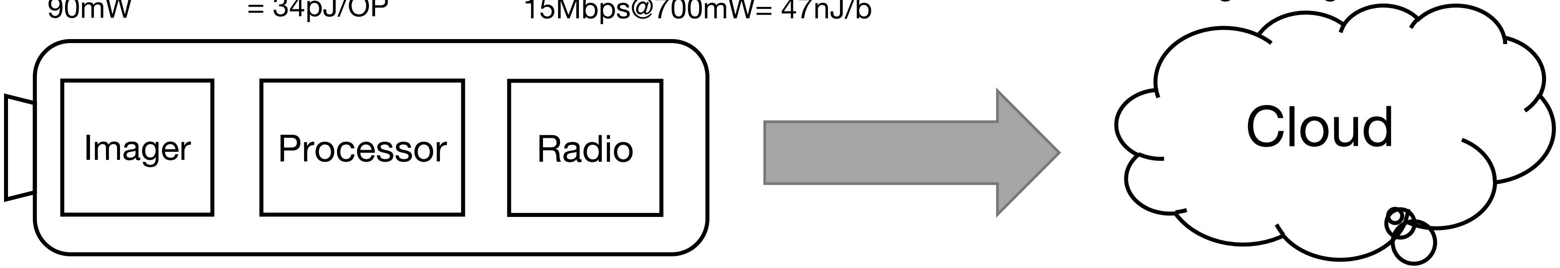
Omnivision
OV2740
90mW

Tegra K1 GPU
290GOPS@10W
= 34pJ/OP

Qualcomm SD810 LTE
>800mW
Atheros 802.11 a/g
15Mbps@700mW= 47nJ/b

Amazon EC2

CPU	c4.large	2x400GFLOPS	$0.1/h
GPU	g2.2xlarge	2.3TFLOPS	$0.65/h



Workload	Deep learning **300GFLOPS** @ 30GFLOPs/frame, 10fps	
Budget	Device power 30% of 10Wh for 10h = 300mW	Cloud cost $10 person/year
Compute power	**9GFLOPS**	**3.5GFLOPS** (GPU) / **8GFLOPS** (CPU)

Huge gap between workload and budget

Outline

- Model compression
- Serving System

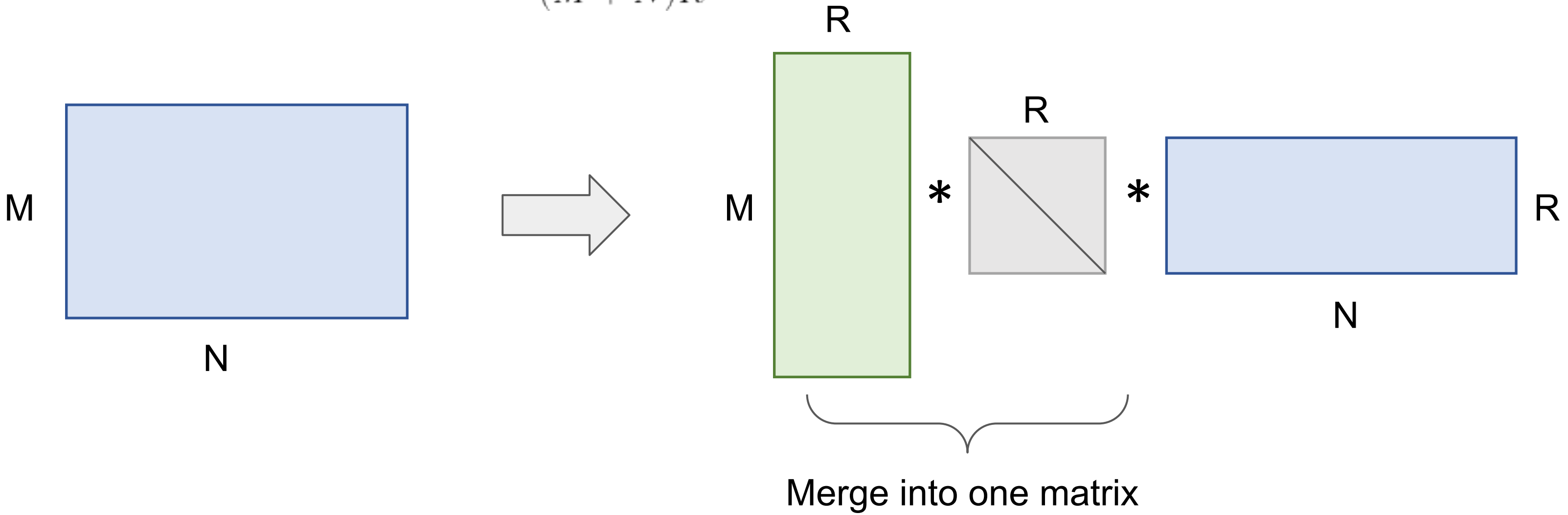
Model Compression

- Tensor decomposition
- Network pruning
- Quantization
- Smaller model

Matrix decomposition

Fully-connected layer

- Memory reduction: $\frac{MN}{(M+N)R}$
- Computation reduction: $\frac{MN}{(M+N)R}$

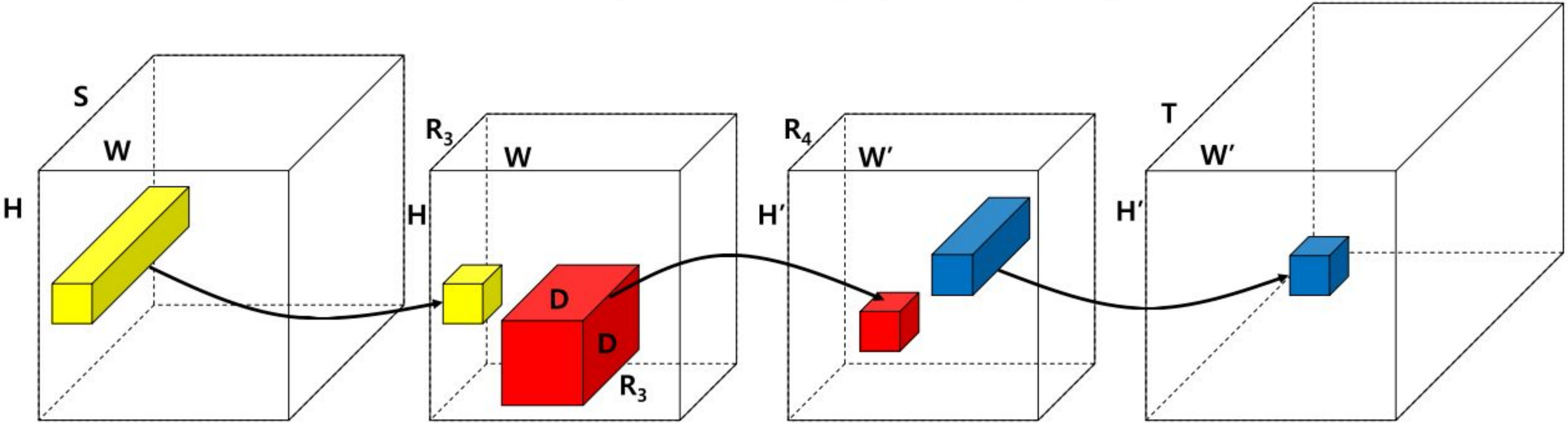


Tensor decomposition

Convolutional layer

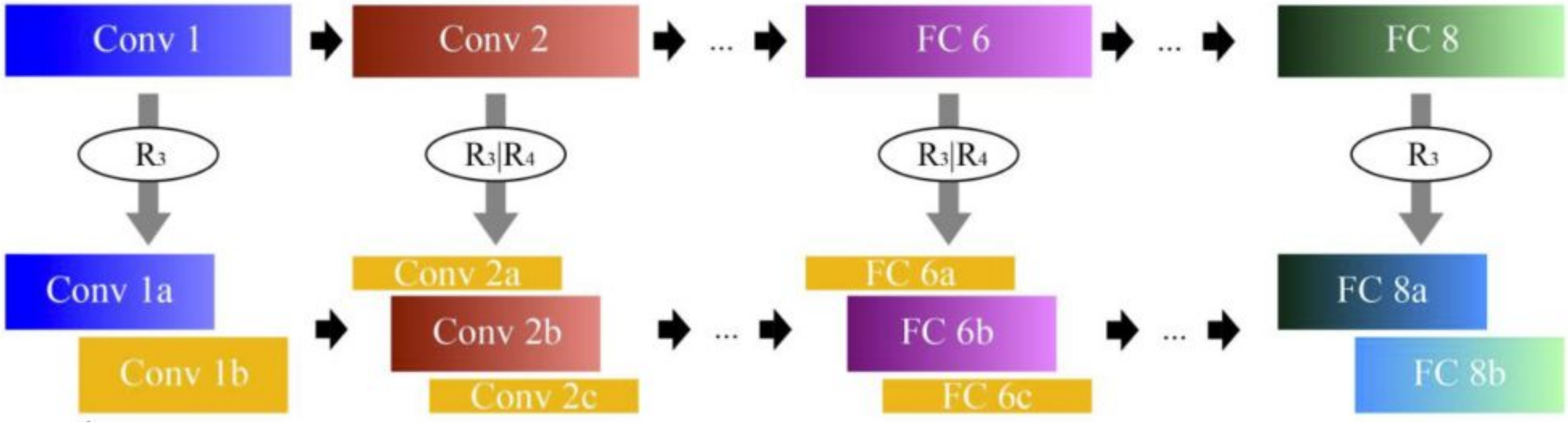
- Memory reduction: $\dfrac{D^2ST}{SR_3 + D^2R_3R_4 + TR_4}$
- Computation reduction: $\dfrac{D^2STH'W'}{SR_3HW + D^2R_3R_4H'W' + TR_4H'W'}$

bounded by ST/R_3R_4

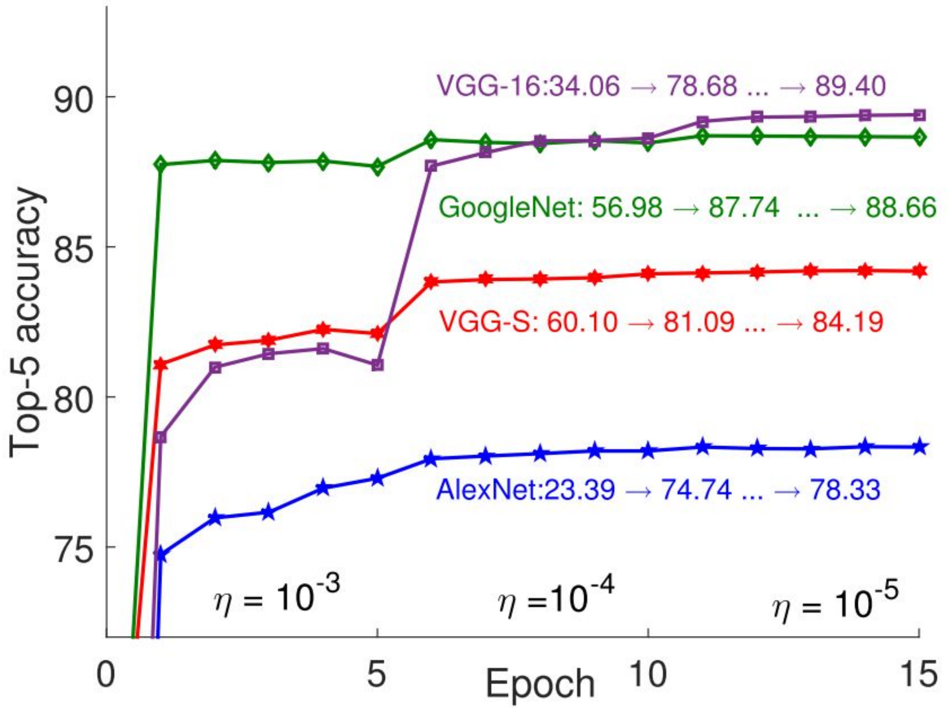


* Kim, Yong-Deok, et al. "Compression of deep convolutional neural networks for fast and low power mobile applications." ICLR (2016).

Decompose the entire model



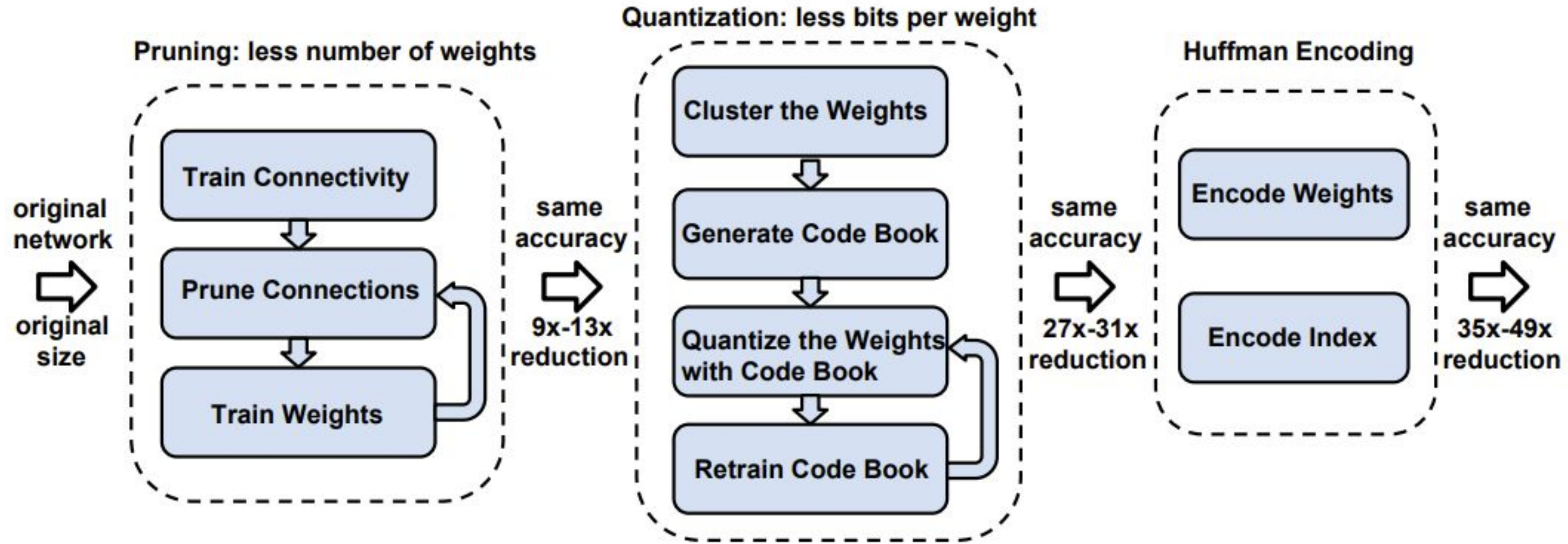
Fine-tuning after decomposition



Accuracy & Latency after Decomposition

Model	Top-5	Weights	FLOPs	S6		Titan X
AlexNet	80.03	61M	725M	117ms	245mJ	0.54ms
*AlexNet**	78.33	11M	272M	43ms	72mJ	0.30ms
(imp.)	(-1.70)	(×5.46)	(×2.67)	(×2.72)	(×3.41)	(×1.81)
VGG-S	84.60	103M	2640M	357ms	825mJ	1.86ms
*VGG-S**	84.05	14M	549M	97ms	193mJ	0.92ms
(imp.)	(-0.55)	(×7.40)	(×4.80)	(×3.68)	(×4.26)	(×2.01)
GoogLeNet	88.90	6.9M	1566M	273ms	473mJ	1.83ms
*GoogLeNet**	88.66	4.7M	760M	192ms	296mJ	1.48ms
(imp.)	(-0.24)	(×1.28)	(×2.06)	(×1.42)	(×1.60)	(×1.23)
VGG-16	89.90	138M	15484M	1926ms	4757mJ	10.67ms
*VGG-16**	89.40	127M	3139M	576ms	1346mJ	4.58ms
(imp.)	(-0.50)	(×1.09)	(×4.93)	(×3.34)	(×3.53)	(×2.33)

Network pruning: Deep compression



* Song Han, et al. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." ICLR (2016).

Network pruning: prune the connections

Initialization: $W^{(0)}$ *with* $W^{(0)} \sim N(0, \Sigma)$, *iter* $= 0$.
Hyper-parameter: *threshold*, δ.
Output : $W^{(t)}$.

——— *Train Connectivity* ———

while *not converged* **do**
    $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
    $t = t + 1$;
end

——— *Prune Connections* ———

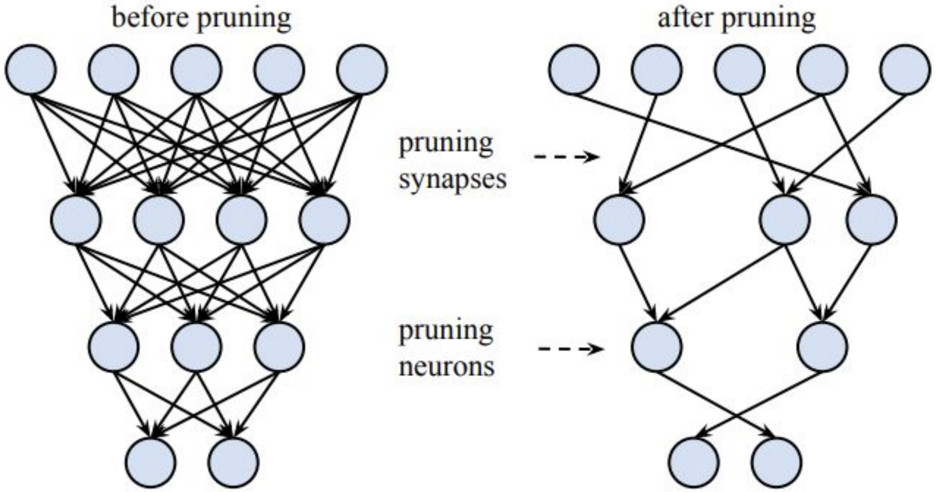
// *initialize the mask by thresholding the weights.*
$Mask = \mathbb{1}(|W| > threshold)$;
$W = W \cdot Mask$;

——— *Retrain Weights* ———

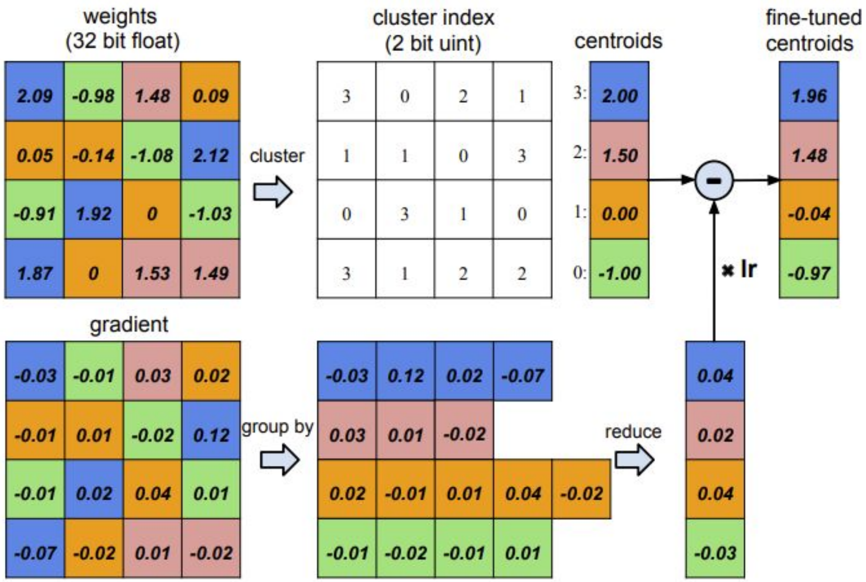
while *not converged* **do**
    $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
    $W^{(t)} = W^{(t)} \cdot Mask$;
    $t = t + 1$;
end

——— *Iterative Pruning* ———

$threshold = threshold + \delta[iter++]$;
goto *Pruning Connections;*



Network pruning: weight sharing



Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom).

1. Use k-means clustering to identify the shared weights for each layer of a trained network. Minimize

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|^2$$

2. Finetune the neural network using shared weights.

Network pruning: accuracy

Table 1: The compression pipeline can save $35\times$ to $49\times$ parameter storage with no loss of accuracy.

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40**$\times$
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39**$\times$
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35**$\times$
VGG-16 Ref	31.50%	11.32%	552 MB	
VGG-16 Compressed	31.17%	10.91%	**11.3 MB**	**49**$\times$

Network pruning: accuracy vs compression

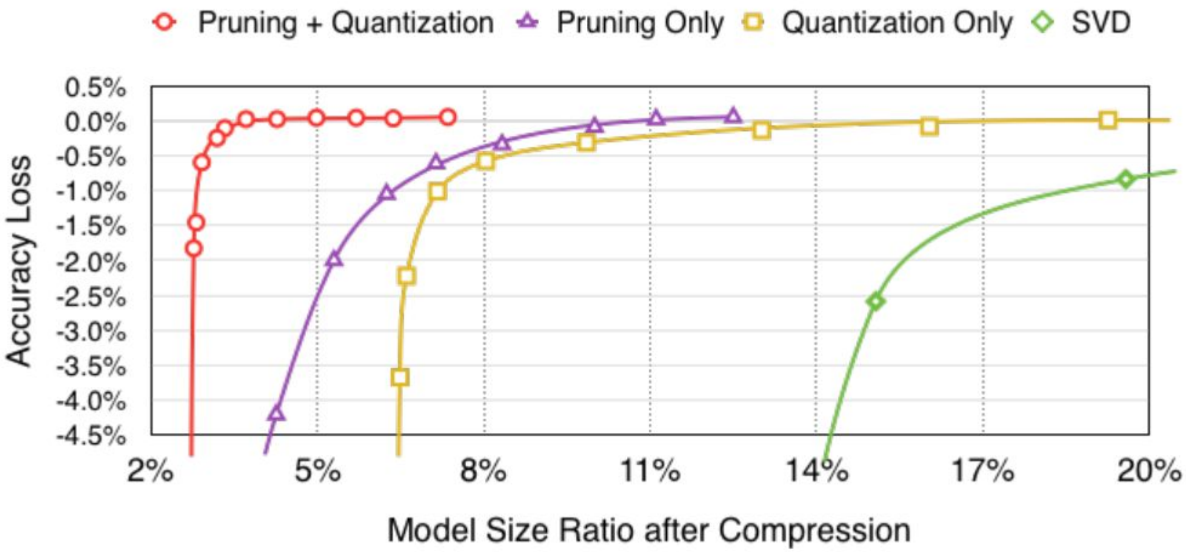


Figure 6: Accuracy v.s. compression rate under different compression methods. Pruning and quantization works best when combined.

Quantization: XNOR-Net

	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Real-Value Weights: 0.12 -1.2 ... 0.41 / -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs: 0.11 -0.21 ... -0.34 / -0.25 0.61 ... 0.52; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs: 1 -1 ... -1 / -1 1 ... 1; Binary Weights: 1 -1 ... 1 / -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

* Mohammad Rastegari, et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks." ECCV (2016).

Quantization: binary weights

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:
for $l = 1$ to L **do**
  for k^{th} filter in l^{th} layer **do**
    $\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$
    $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}^t_{lk})$
    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$
$\hat{\mathbf{Y}} = \textbf{BinaryForward}(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 1 or 11
$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} = \textbf{BinaryBackward}(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$
$\mathcal{W}^{t+1} = \textbf{UpdateParameters}(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (*e.g.*,SGD or ADAM)
$\eta^{t+1} = \textbf{UpdateLearningrate}(\eta^t, t)$ // Any learning rate scheduling function

Quantization: binary input and weights

Binary Dot Product: To approximate the dot product between $\mathbf{X}, \mathbf{W} \in \mathbb{R}^n$ such that $\mathbf{X}^\top \mathbf{W} \approx \beta \mathbf{H}^\top \alpha \mathbf{B}$, where $\mathbf{H}, \mathbf{B} \in \{+1, -1\}^n$ and $\beta, \alpha \in \mathbb{R}^+$, we solve the following optimization:

$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha, \mathbf{B}, \beta, \mathbf{H}}{\operatorname{argmin}} \|\mathbf{X} \odot \mathbf{W} - \beta\alpha\mathbf{H} \odot \mathbf{B}\| \tag{7}$$

$$\mathbf{C}^* = \operatorname{sign}(\mathbf{Y}) = \operatorname{sign}(\mathbf{X}) \odot \operatorname{sign}(\mathbf{W}) = \mathbf{H}^* \odot \mathbf{B}^*$$

$$\gamma^* = \frac{\sum |\mathbf{Y}_i|}{n} = \frac{\sum |\mathbf{X}_i||\mathbf{W}_i|}{n} \approx \left(\frac{1}{n}\|\mathbf{X}\|_{\ell 1}\right)\left(\frac{1}{n}\|\mathbf{W}\|_{\ell 1}\right) = \beta^* \alpha^*$$

Quantization: accuracy

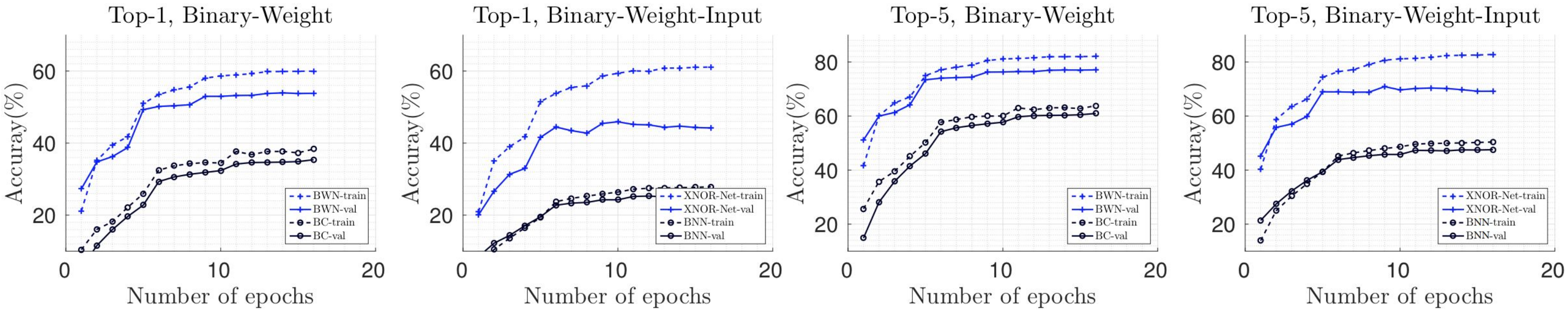
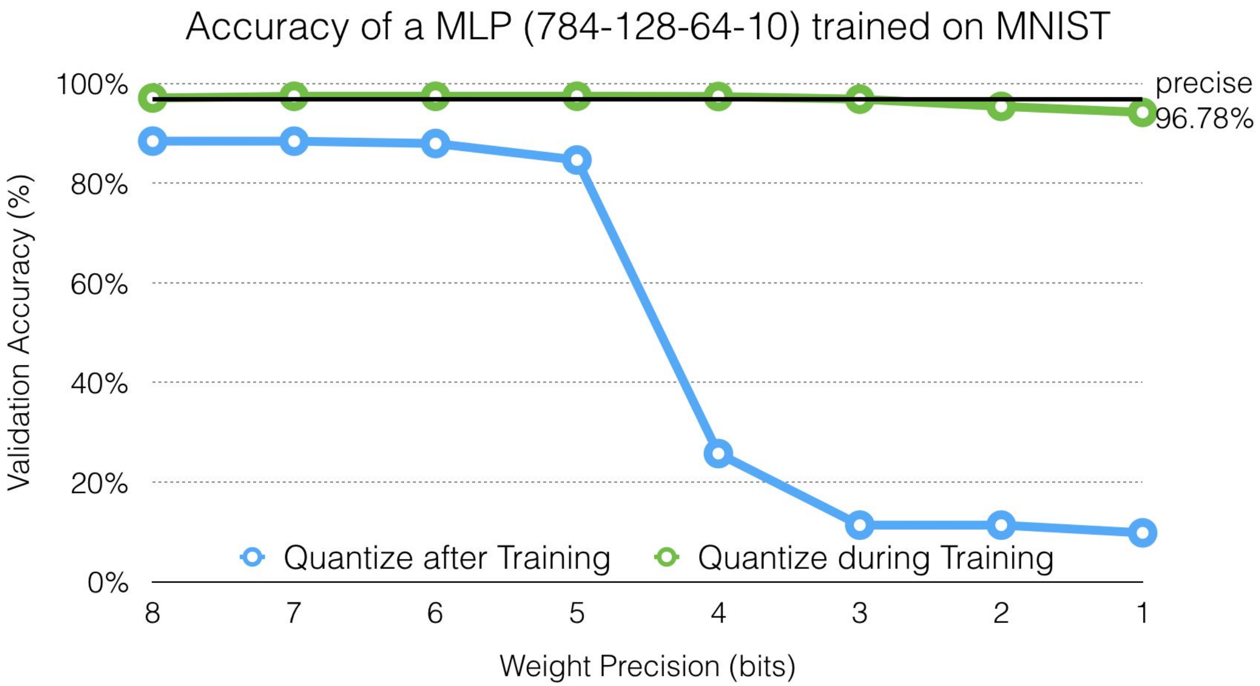
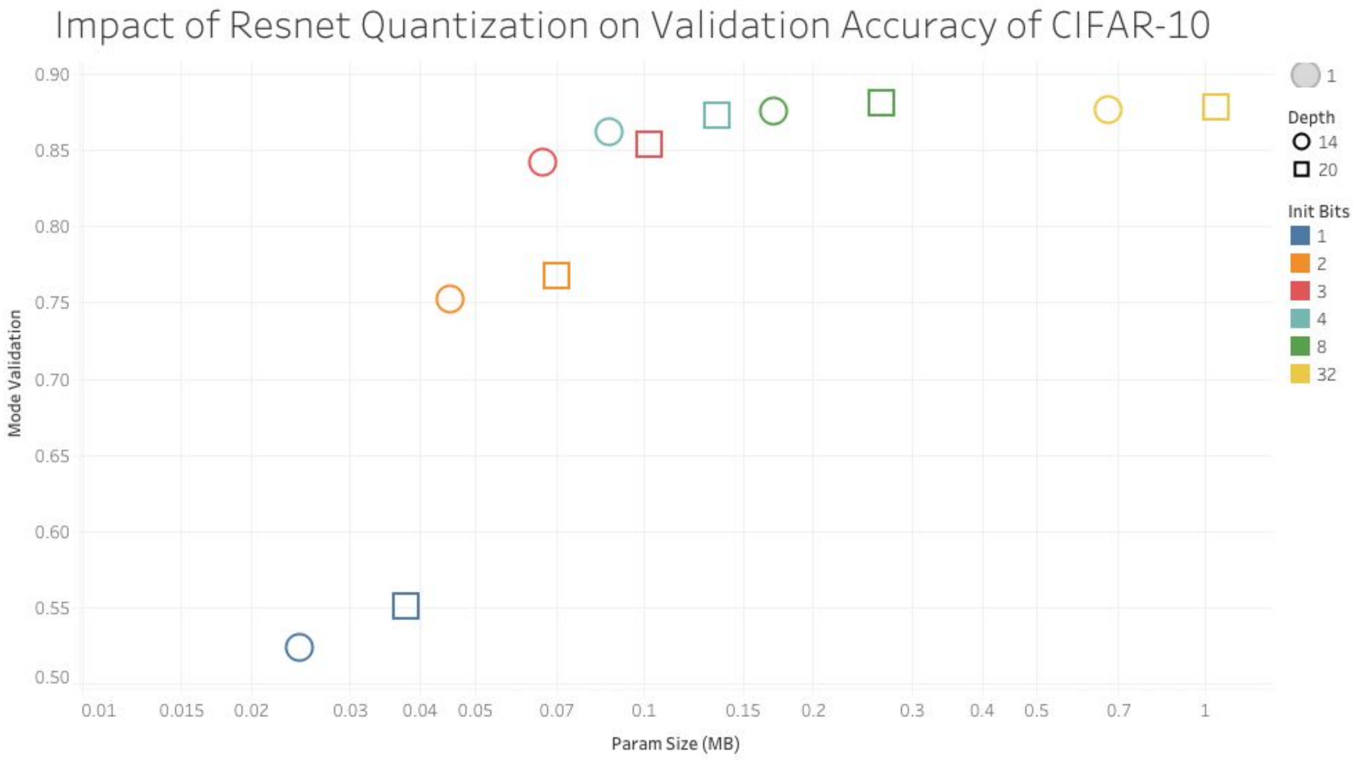


Fig. 5: This figure compares the imagenet classification accuracy on Top-1 and Top-5 across training epochs. Our approaches BWN and XNOR-Net outperform BinaryConnect(BC) and BinaryNet(BNN) in all the epochs by large margin($\sim$17%).

Quantization



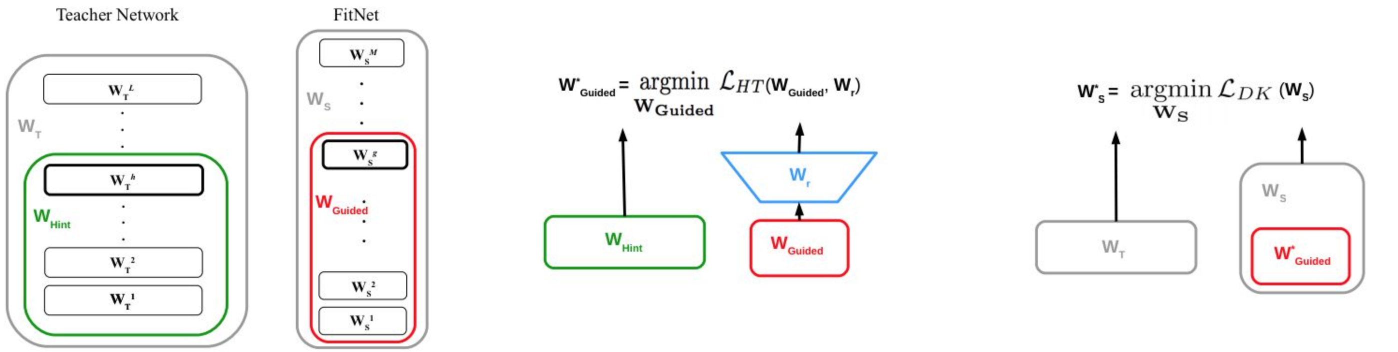
Quantization



Param Size (MB) vs. Mode Validation. Color shows details about sum of Init Bits. Size shows sum of Width Factor. Shape shows details about sum of Depth. The data is filtered on Weight Mode, which keeps tanh. The view is filtered on sum of Width Factor, which ranges from 1 to 1.

Smaller model: Knowledge distillation

- Knowledge distillation: use a teacher model (large model) to train a student model (small model)



(a) Teacher and Student Networks (b) Hints Training (c) Knowledge Distillation

$$\mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r}) = \frac{1}{2}||u_h(\mathbf{x}; \mathbf{W_{Hint}}) - r(v_g(\mathbf{x}; \mathbf{W_{Guided}}); \mathbf{W_r})||^2,$$

* Romero, Adriana, et al. "Fitnets: Hints for thin deep nets." ICLR (2015).

Smaller model: accuracy

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**91.61%**
Teacher	~9M	90.18%
Mimic single	~54M	84.6%
Mimic single	~70M	84.9%
Mimic ensemble	~70M	85.8%
State-of-the-art methods		
Maxout		90.65%
Network in Network		91.2%
Deeply-Supervised Networks		**91.78%**
Deeply-Supervised Networks (19)		88.2%

Table 1: Accuracy on CIFAR-10

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**64.96%**
Teacher	~9M	63.54%
State-of-the-art methods		
Maxout		61.43%
Network in Network		64.32%
Deeply-Supervised Networks		**65.43%**

Table 2: Accuracy on CIFAR-100

Discussion

What are the implications of these model compression techniques for serving?

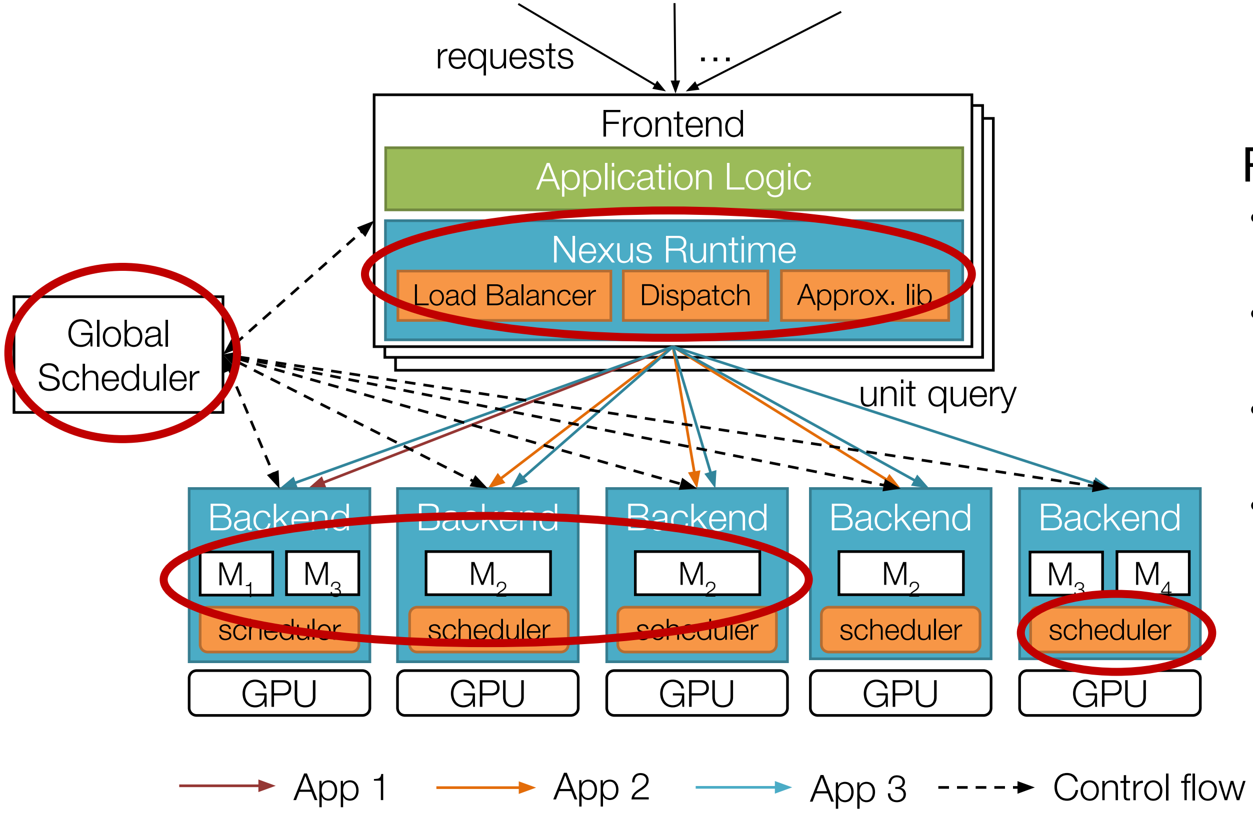
- Specialized hardware for sparse models
  - Song Han, et al. “EIE: Efficient Inference Engine on Compressed Deep Neural Network.” ISCA 2016
- Accuracy and resource trade-off
  - Han, Seungyeop, et al. "MCDNN: An Approximation-Based Execution Framework for Deep Stream Processing Under Resource Constraints." MobiSys (2016).

Serving system

Serving system

- Goals:
  - High flexibility for writing applications
  - High efficiency on GPUs
  - Satisfy latency SLA
- Challenges
  - Provide common abstraction for different frameworks
  - Achieve high efficiency
    - Sub-second latency SLA that limits the batch size
    - Model optimization and multi-tenancy causes long tail

Nexus: efficient neural network serving system



Remarks

- Frontend runtime library allows arbitrary app logic
- Packing models to achieve higher utilization
- A GPU scheduler allows new batching primitives
- A batch-aware global scheduler allocates GPU cycles for each model

Flexibility: Application runtime

```
class ModelHandler:
  # return output future
  def Execute(input)

class AppBase:
  # return ModelHandler
  def GetModelHandler(framework, model, version, latency_sla)
  # Load models during setup time, implemented by developer
  def Setup()
  # Process requests, implemented by developer
  def Process(request)
```

Async RPC, execute the model remotely

Send load model request to global scheduler

Application example: Face recognition

```
class FaceRecApp(AppBase):
  def Setup(self):
    self.m1 = self.GetModelHandler("caffe", "vgg_face", 1, 100)
    self.m2 = self.GetModelHandler("mxnet", "age_net", 1, 100)

  def Process(self, request):
    ret1 = self.m1.Execute(request.image)
    ret2 = self.m2.Execute(request.image)
    return Reply(request.user_id, ret1["name"], ret2["age"])
```

Load model from different framework

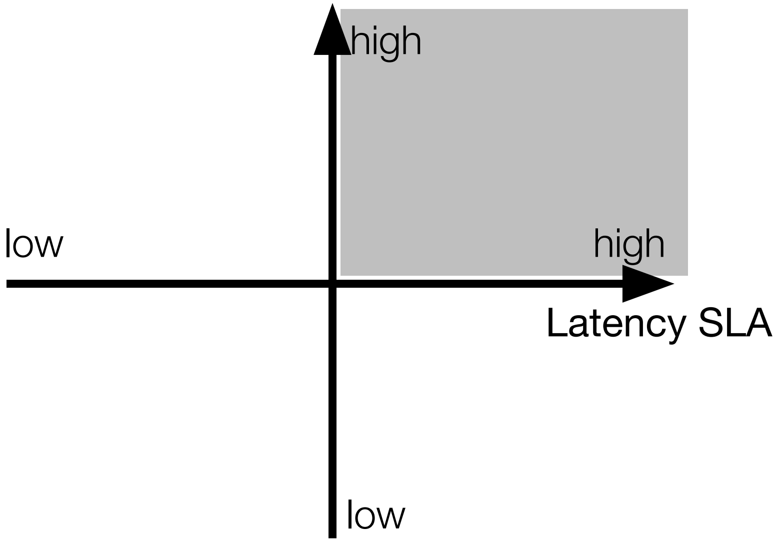
Execute concurrently on remote GPUs

Force to synchronize when accessing future data

Application example: Traffic Analysis

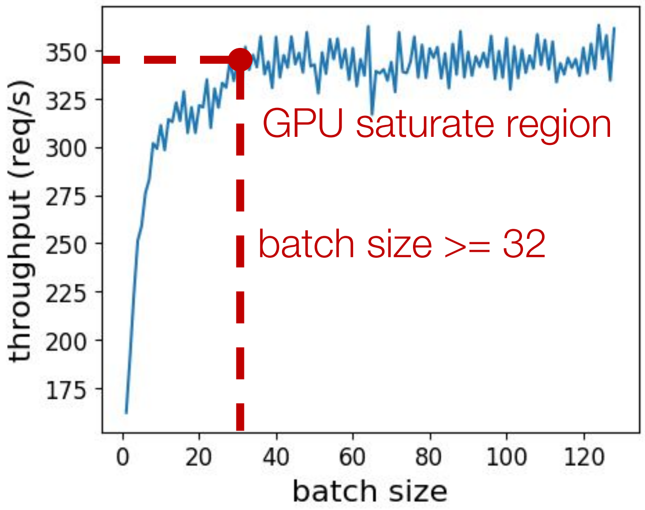
```
class TrafficApp(AppBase):
  def Setup(self):
    self.det = self.GetModelHandler(“darknet”, “yolo9000”, 1, 300)
    self.r1 = self.GetModelHandler(“caffe”, “vgg_face”, 1, 150)
    self.r2 = self.GetModelHandler(“caffe”,“googlenet_car”, 1, 150)

  def Process(self, request):
    persons, cars = [], []
    for obj in self.det.Execute(request.image):
      if obj[“class”] == “person”:
        persons.append(self.r1.Execute(request.image[obj[“rect”]])
      elif obj[“class”] == “car”:
        cars.append(self.r2.Execute(request.image[obj[“rect”]])
    return Reply(request.user_id, persons, cars)
```

High Efficiency



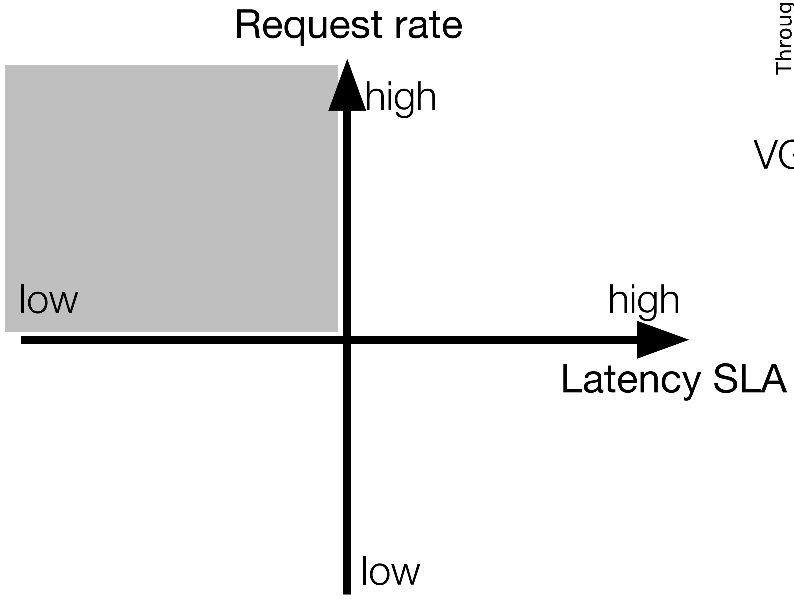
Workload characteristic

- For high request rate, high latency SLA workload, saturate GPU efficiency by using large batch size

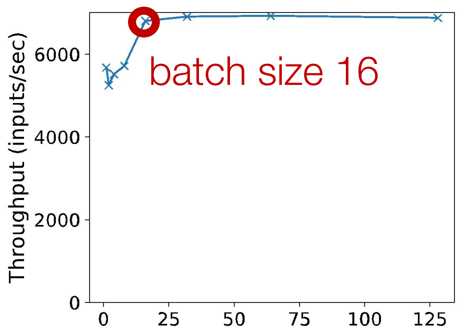


VGG16 throughput vs batch size

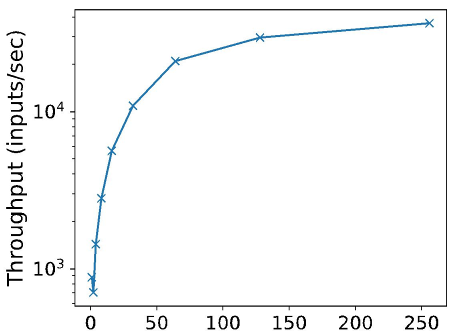
High Efficiency



Workload characteristic

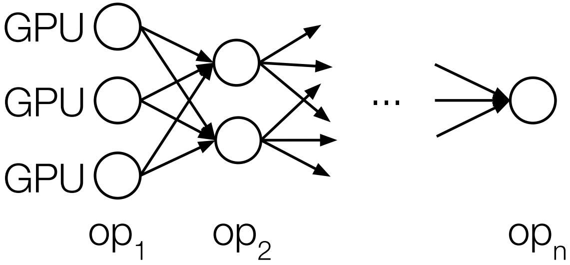


VGG16 conv2_1 (winograd)



VGG16 fc6

- Suppose we can choose a different batch size for each op (layer), and allocate dedicated GPUs for each op.



Split Batching

- Optimization problem

$$\max_{b_1 \dots b_n} \frac{\max_{\mathrm{j}} tp_j(b_j)}{\sum_i \max_{\mathrm{j}} tp_j(b_j)\,/tp_i(b_i)}$$

equivalent to

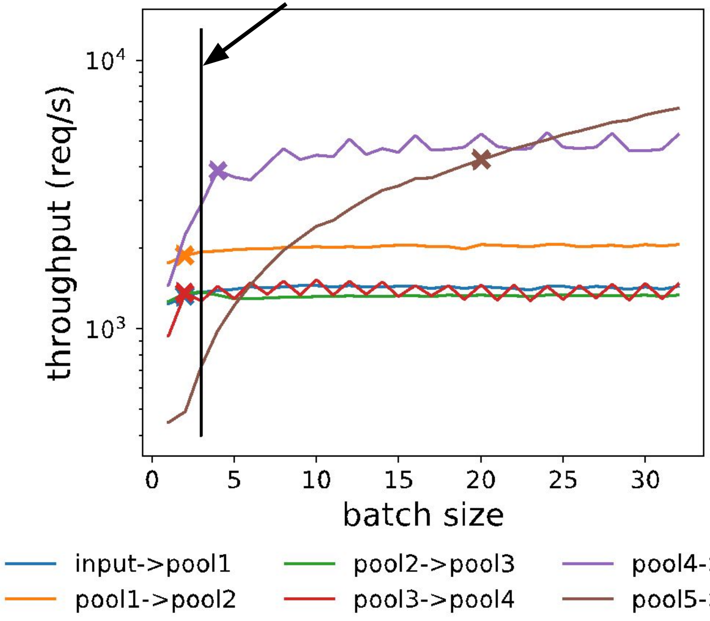
$$\min_{b_1 \dots b_n} \sum_i 1/tp_i(b_i)$$

such that

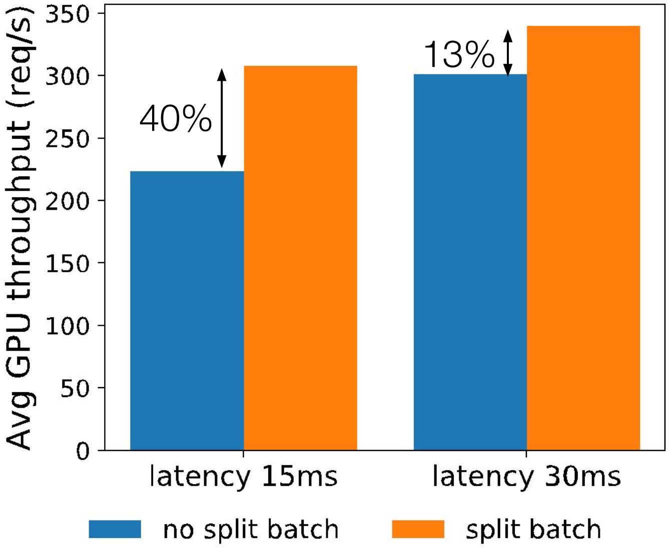
$$\sum_i lat_i(b_i) + \sum_{b_i \neq b_{i+1}} overhead(out_i * b_i) \leq latency\_sla$$

Split Batching

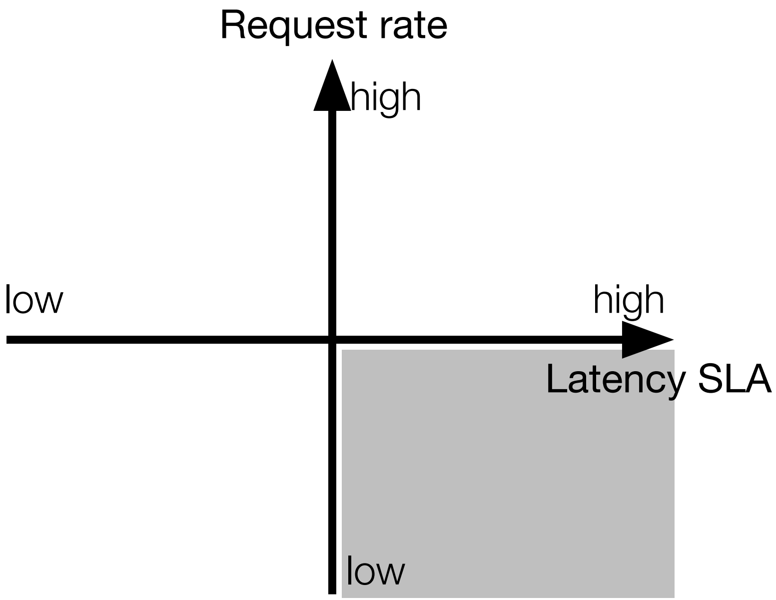
Batch size 3 for entire model



VGG16 optimal batch size for each segment for latency SLA 15ms

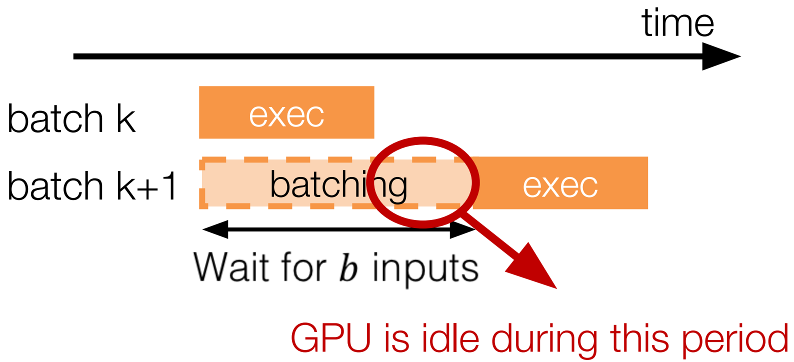


High Efficiency



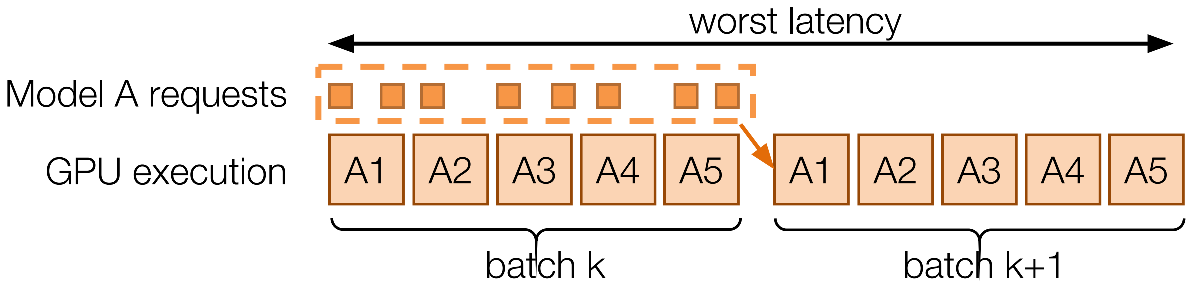
Workl

- This type of workload cannot saturate GPU in temporal domain
- Suppose the optimal batch size is b under latency SLA



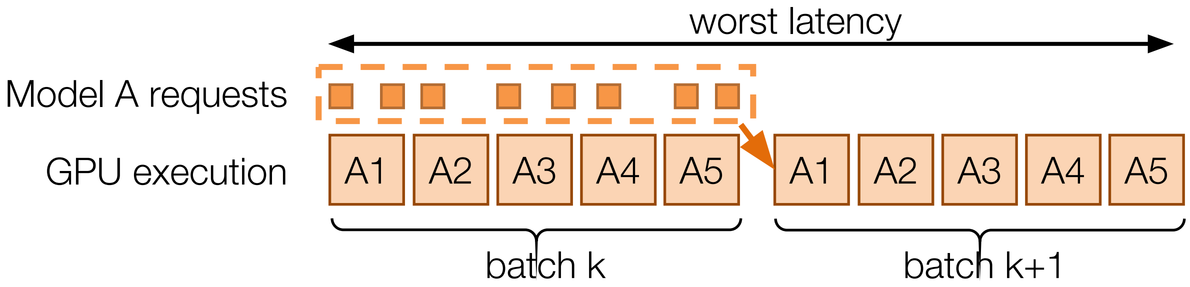
Execute multiple models on one GPU

Execute multiple models on a single GPU

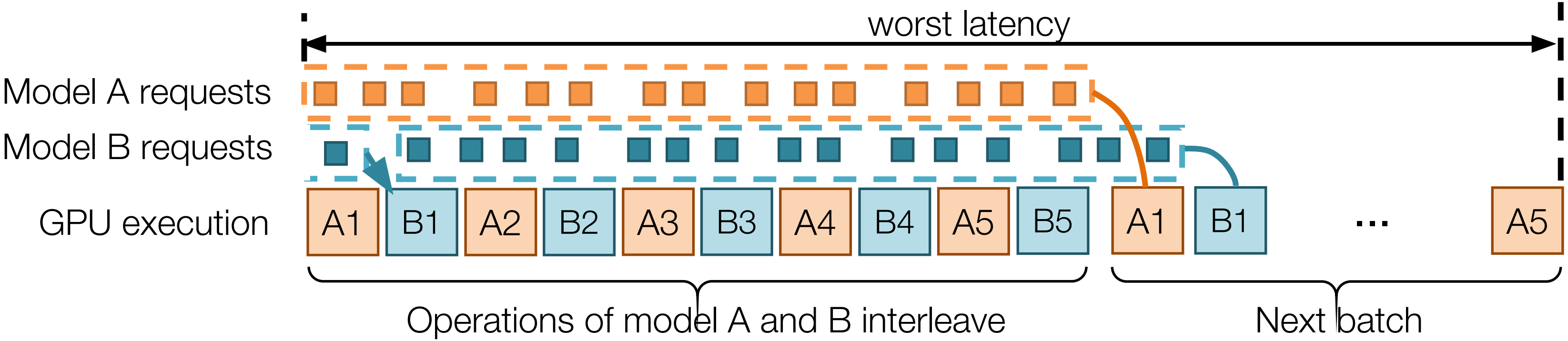


(a) Single model execution: worst latency is $2\mathrm{lat}_A$

Execute multiple models on a single GPU

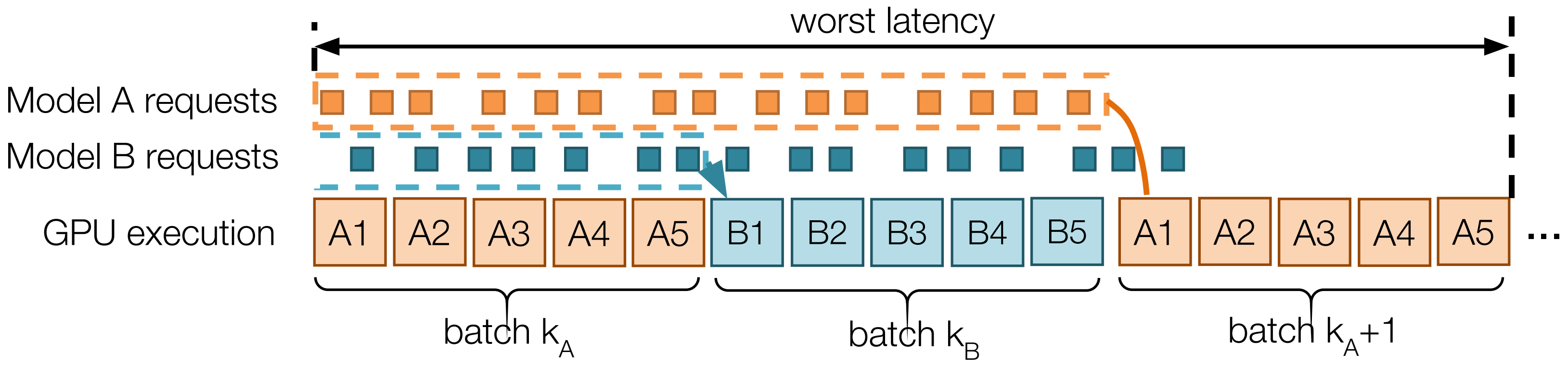


(a) Single model execution: worst latency is $2\mathrm{lat}_A$



(b) Execute multiple models concurrently:
worst latency for model A and B is $2(\mathrm{lat}_A + \mathrm{lat}_B)$

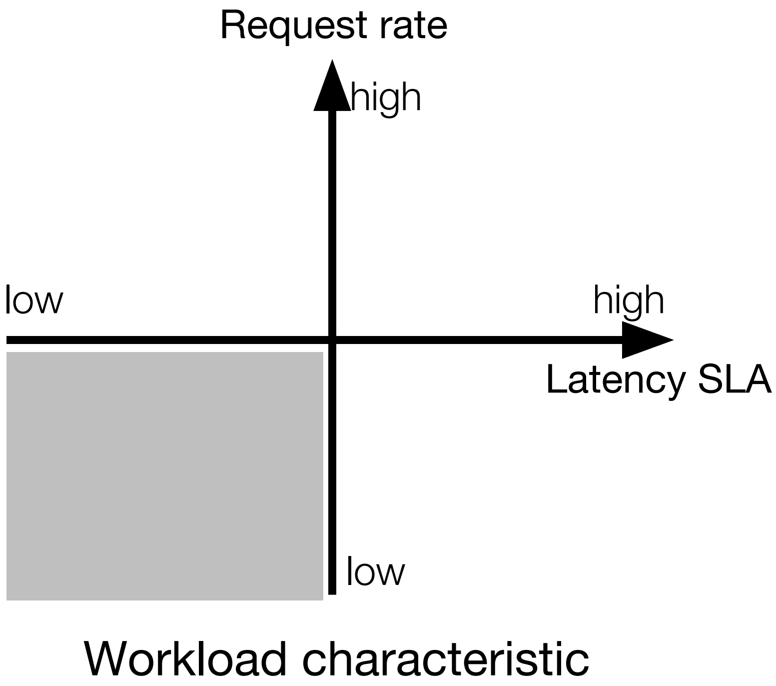
Execute multiple models on a single GPU



(c) Execute multiple models in round-robin fashion:
worst latency for model A is $2\mathrm{lat}_A + \mathrm{lat}_B$
worst latency for model B is $\mathrm{lat}_A + 2\mathrm{lat}_B$

Use larger batch size as latency is reduced and predictive

High Efficiency

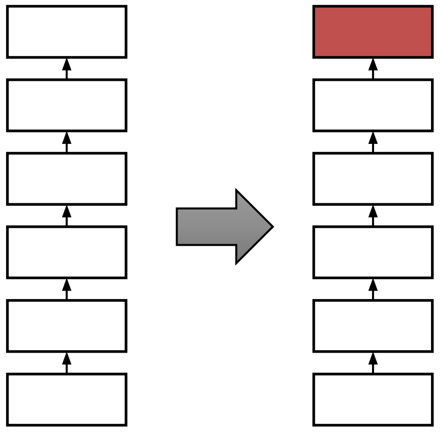


Solution depends

- If saturate GPU in temporal domain due to low latency: allocate dedicated GPU(s)
- If not: can use multi-batching to share GPU cycles with other models

Prefix batching for model specialization

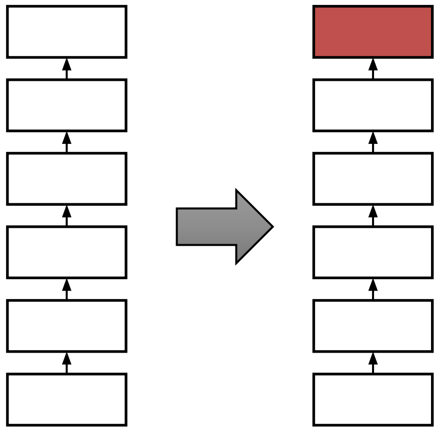
- Specialization (long-term / short-term) re-trains last a few layers



A new model

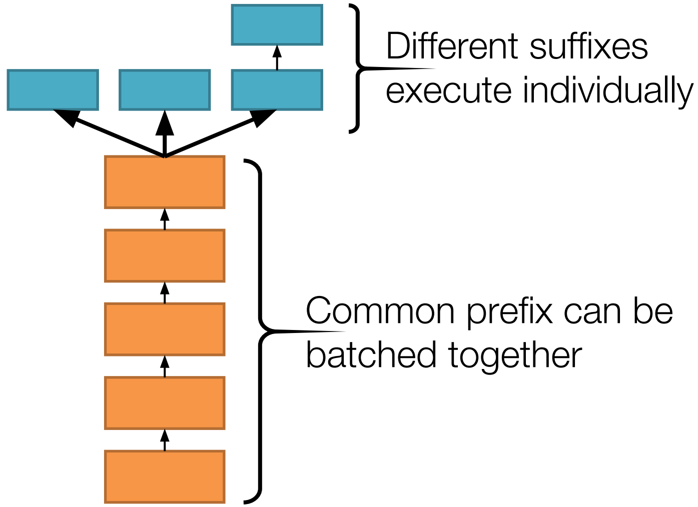
Prefix batching for model specialization

- Specialization (long-term / short-term) re-trains last a few layers



A new model

- Prefix batching allows batch execution for common prefix

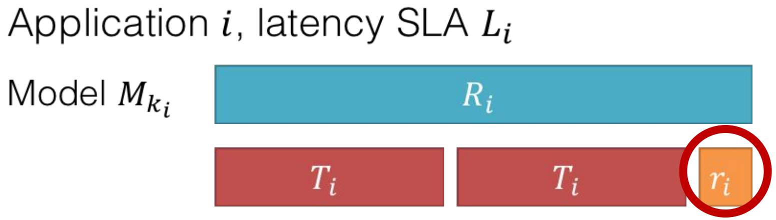


Meet Latency SLA: Global scheduler

- First apply split batching and prefix batching if possible
- Multi-batching: bin-packing problem to pack models into GPUs
- Bin-packing optimization goal
  - Minimize the resource usage (number of GPUs)
- Constraint
  - Requests have to be served within latency SLA
- Degrees of freedom
  - Split workloads into smaller tasks
  - Change the batch size

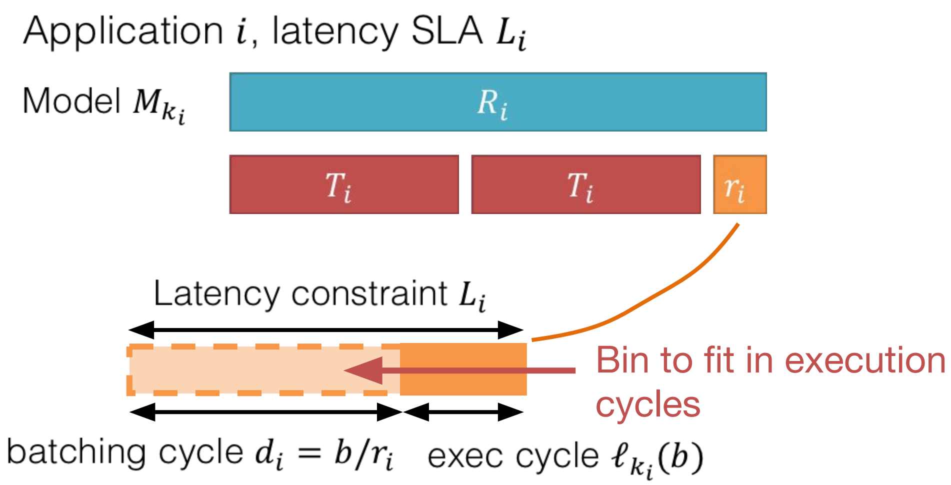
Best-fit decreasing algorithms

1. For each workload, T_i is the max throughput that can be achieved on a GPU within latency SLA, and allocate dedicated GPUs



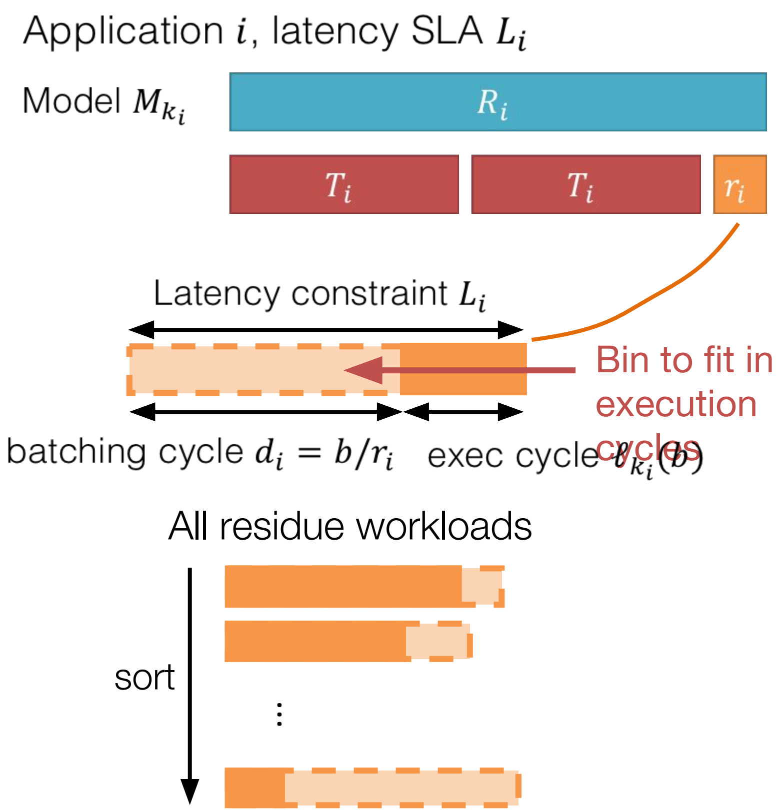
Best-fit decreasing algorithms

1. For each workload, T_i is the max throughput that can be achieved on a GPU within latency SLA, and allocate dedicated GPUs
2. For each residue workload, split latency SLA into batching cycle and exec cycle



Best-fit decreasing algorithms

1. For each workload, T_i is the max throughput that can be achieved on a GPU within latency SLA, and allocate dedicated GPUs
2. For each residue workload, split latency SLA into batching cycle and exec cycle
3. Sort all residue workloads by occupancy $\ell_{k_i}(B_i)/d_i$, and merge them by best-fit



Reference

- Kim, Yong-Deok, et al. "Compression of deep convolutional neural networks for fast and low power mobile applications." ICLR (2016).
- Song Han, et al. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." ICLR (2016).
- Romero, Adriana, et al. "Fitnets: Hints for thin deep nets." ICLR (2015).
- Mohammad Rastegari, et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks." ECCV (2016).
- Crankshaw, Daniel, et al. "Clipper: A Low-Latency Online Prediction Serving System." NSDI (2017).