DeepReDuce: ReLU Reduction for Fast Private Inference

Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, Brandon Reagen

{nj2049,zg451,sg175,bjr5 }@nyu.edu

Department of Electrical and Computer Engineering

New York University

February 16, 2021

Outline

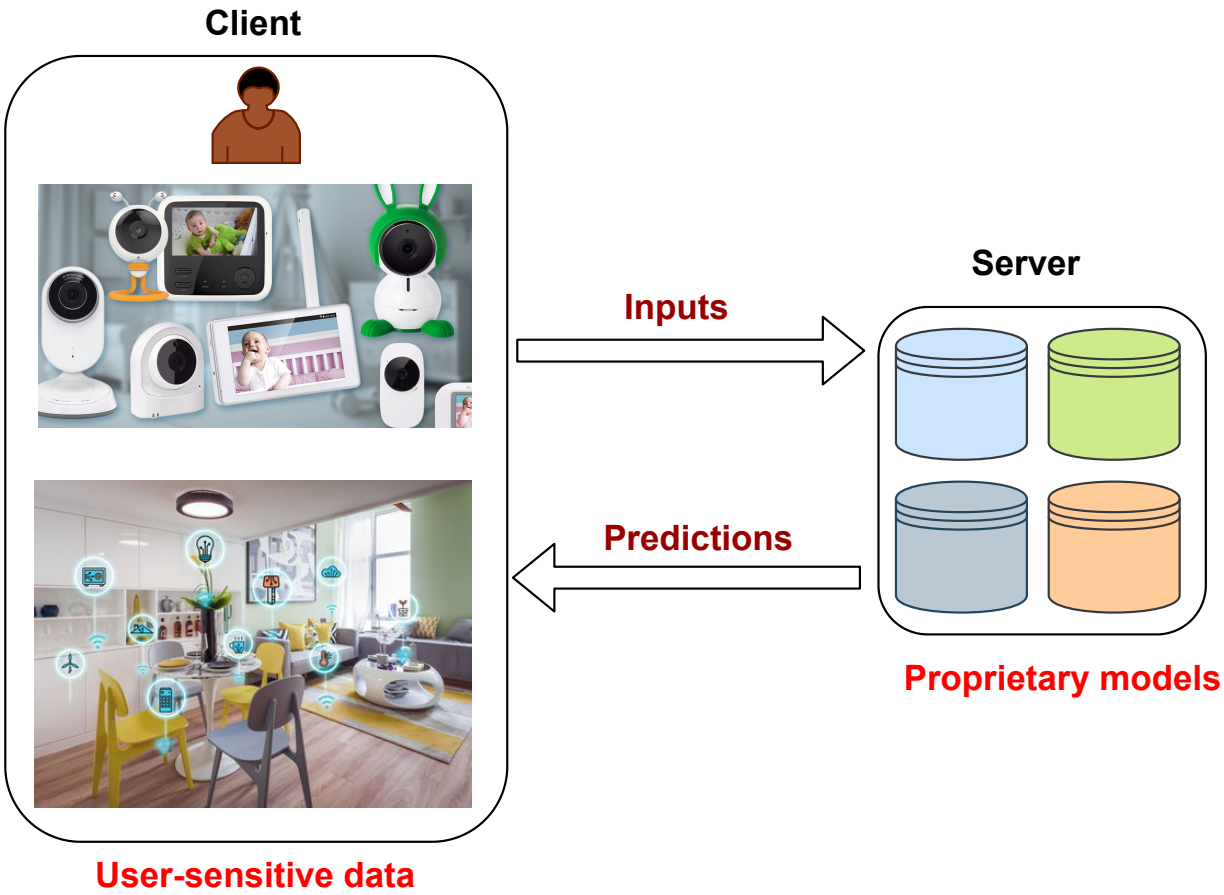
- Privacy-preserving Computation in Retrospective:
  - An overview and introduction
  - Motivation and challenges
  - Finding the gap
- Heterogeneity of ReLUs
- Proposed Method: DeepReDuce
  - Exploiting the heterogeneity of ReLUs
  - Gradual ReLU reduction upto 32×
- Experimental Results
- Conclusion and Future Work

Motivation and Challenges

"We need AI systems that make timely and safe decisions in unpredictable environments,that are robust against sophisticated adversaries, and **that can process ever increasing amounts of data across organizations and individuals without compromising confidentiality**"

Ref: [Stoica et al., A Berkeley View of Systems Challenges for AI]

- User-data is privacy-sensitive
- Trained model is intellectual propriety (IP) of service provider
  - Model weights can hint the training data
  - Training a model can take 100s of GPU-hours



Motivation and Challenges

Private Inference goal: Can client and server collaboratively perform computations without revealing (learning) each other's data except the outcome of computations.

- Client-side inference
  - Model weights and architecture are exposed to user.
  - Model-stealing (IP infringement), and it can leak training data
- Server-side inference
  - User's private data is exposed to server.

Solution?

Two-party computation!

Aim: User *should not learn* anything about server's (private) model and server *should learn nothing* about user's (private) data

Prior Work on Secure Computation

Pitfalls of Earlier Work on Secure Computation

Two-party Computation using Garbled Circuit

(+) Light-weight and computationally efficient
(-) Inefficient communication (data x'fer $\propto$ to boolean circuit size)
(-) Do not support native arithmetic (e.g., matrix-vector) computations; inherently unsuitable for ML computations.

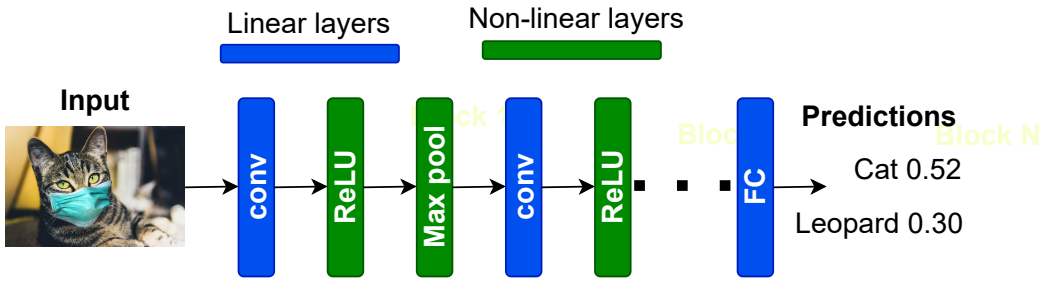
Fully Homomorphic Encryption

(+) Efficient Communication (data x'fer $\propto$ to input image size)
(+) Support native arithmetic computation; amenable for ML computation.
(-) Huge computation overhead at server side (proportional to degree of polynomials)

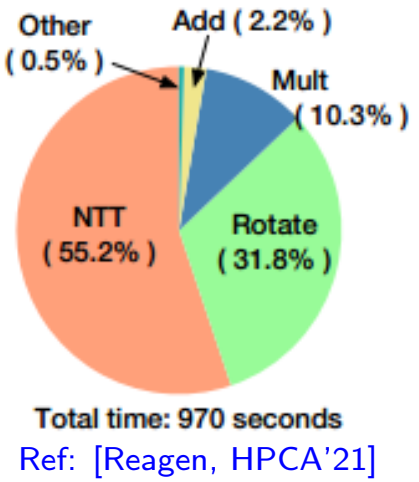
Can we get the best of both worlds?

Gazelle: A Hybrid Approach for Private Inference

Gazelle [USENIX'18] uses **homomorphic encryption** for linear layers and **Garbled circuit** for non-linear layers computation



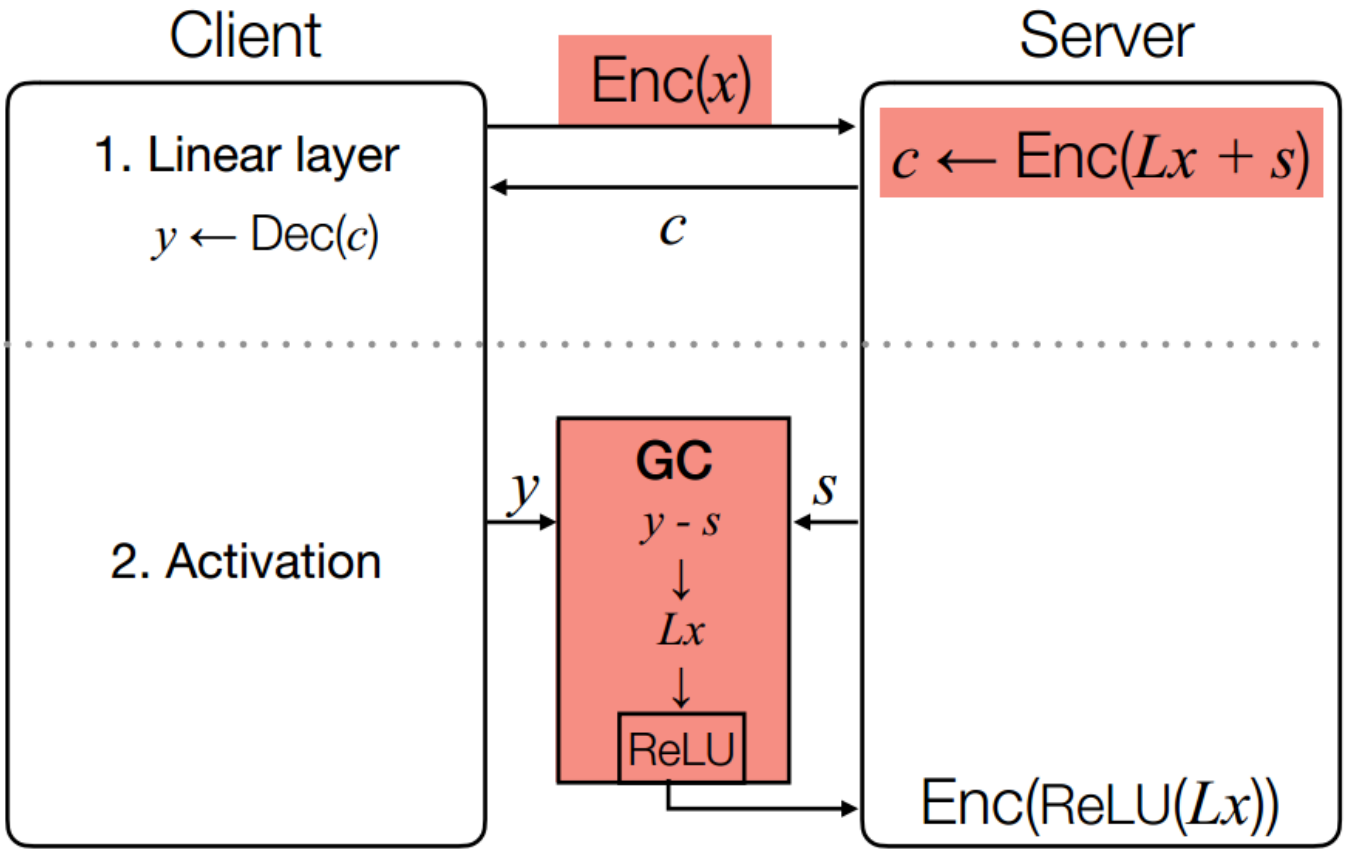
- ML computations heavily involve linear-algebra kernels for computation in linear layers
  - conv layers use Matrix-Matrix and FC layers use Matrix-Vector computation
- HE computation spend most of the time in **NTT**.



Ref: [Reagen, HPCA'21]

Gazelle optimized the linear-algebra kernels for faster computations in linear layers and made **NTT** division free

Performance Overhead in Gazelle

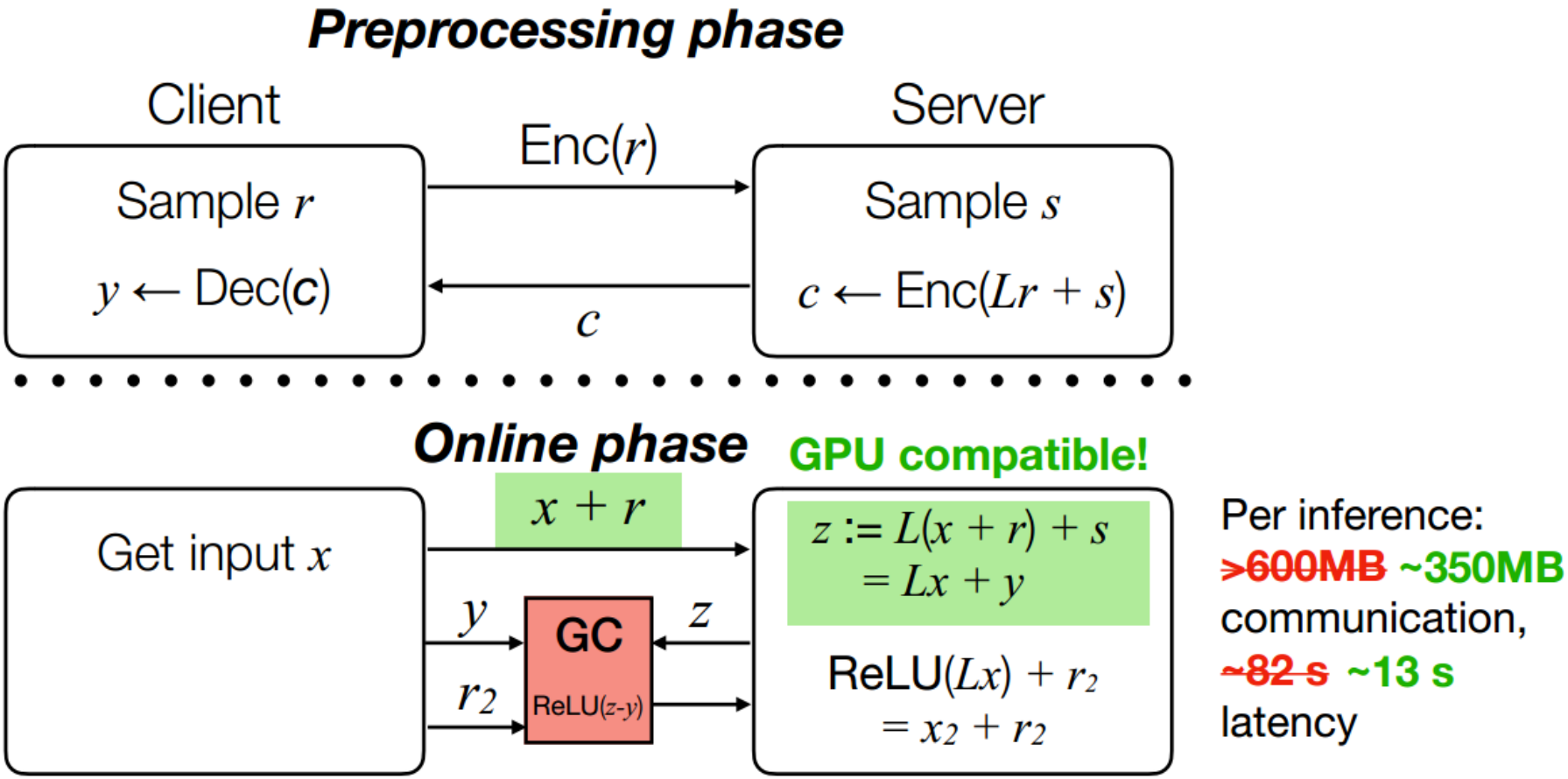


For ResNet-32, per inference: **~600MB** communication, and **~82 sec** latency.

Ref: [DELPHI, USENIX'20]

Online encryption for linear layers' computation and *lack of GPU support* for HE libraries makes linear layer computation very-slow.

DELPHI: Preprocessing and Online Phases

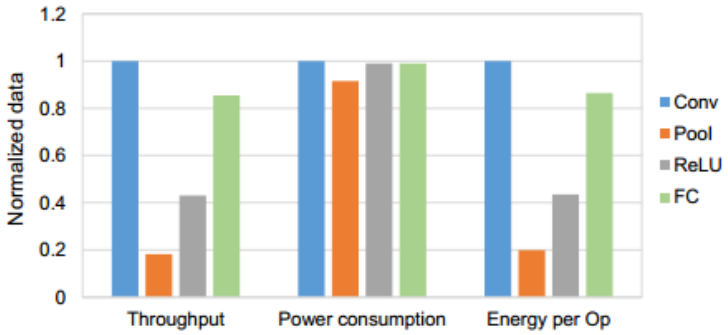


Ref: [DELPHI, USENIX'20]

DELPHI performs *most* of the linear-layer computations offline

Operators Cost in Plaintexts vs. Ciphertexts

Plaintexts operator cost



Ref: Lai+, MLSys'19

- Computation cost of ReLUs is 0.4× of conv.

Ciphertexts operator cost

Operators	Latency (μs) Preproc.	Latency (μs) online	Communication (KB) Preproc.	Communication (KB) online
ReLU	154.9 296×	85.3 12655×	17.5 3889×	2.048 72624×
Quad	6 11.45×	0.03 4.45×	0.152 33.78×	0.008 283.7×
Conv	0.524 1×	6.77×10^{-3} 1×	4.5×10^{-3} 1×	2.82×10^{-5} 1×

Ref: DELPHI, USENIX'20

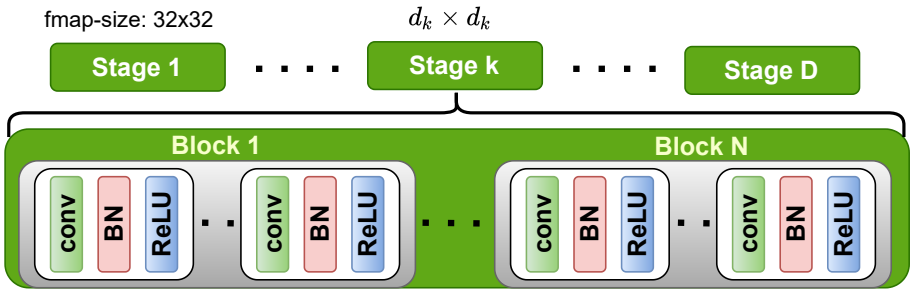
- Computation cost of ReLUs is $\approx 10^5\times$ higher than conv .

Quad activation is cheaper than ReLU, but harder to train deeper networks. Activation and gradients can easily blowup.

The ultimate goal for fast private inference is ReLU reduction!

Challenges for Naively Reducing the ReLU Count

- Every conv layer is followed by a ReLU layer.
- ReLUs can be either enable or disable at the *granularity* of layers in the network.



Huge search space for finding the best ReLU-optimized networks.

- A network with N conv layers would have 2^N possible networks with different ReLU count

  -Deeper networks have larger value of N (≈50-100). For example, ResNet50 would have 2^{50} possible ways of ReLU optimization.

A generalized and efficient ReLU reduction mechanism is need of the day for finding best ReLU-optimized networks!

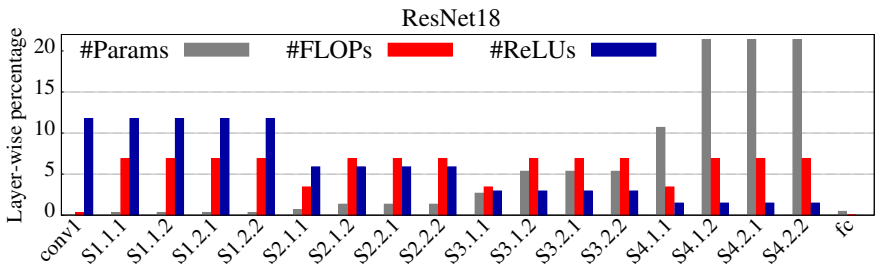
Heterogeneity of ReLUs

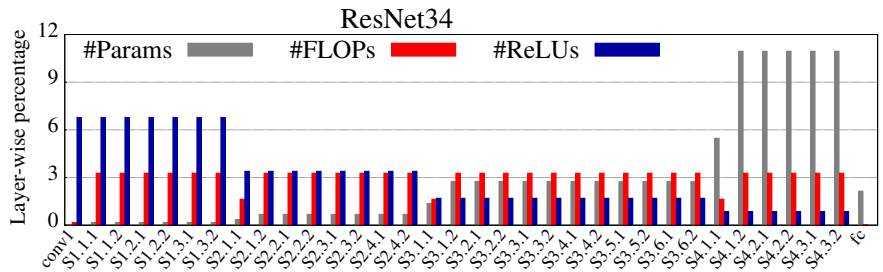
- Distribution of ReLUs in the networks

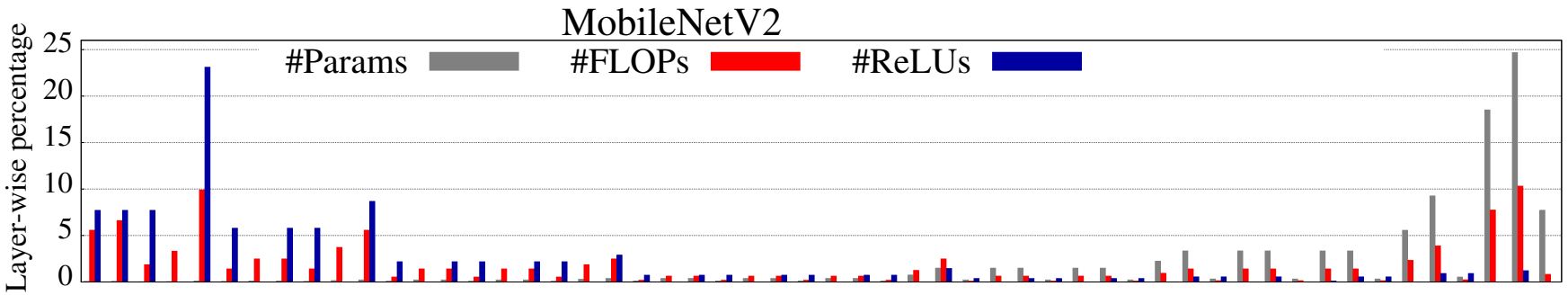
- Are all ReLUs have similar effect on accuracy of the network?

- Accuracy gain from Knowledge Distillation *varies with position* of ReLUs

ReLU Distribution







FLOPs are evenly distributed, ReLUs (parameters) are decreasing (increasing) in the deeper layers.

Observation: Initial layers have disproportionately higher # ReLUs.

Criticality of ReLUs for Network's Accuracy

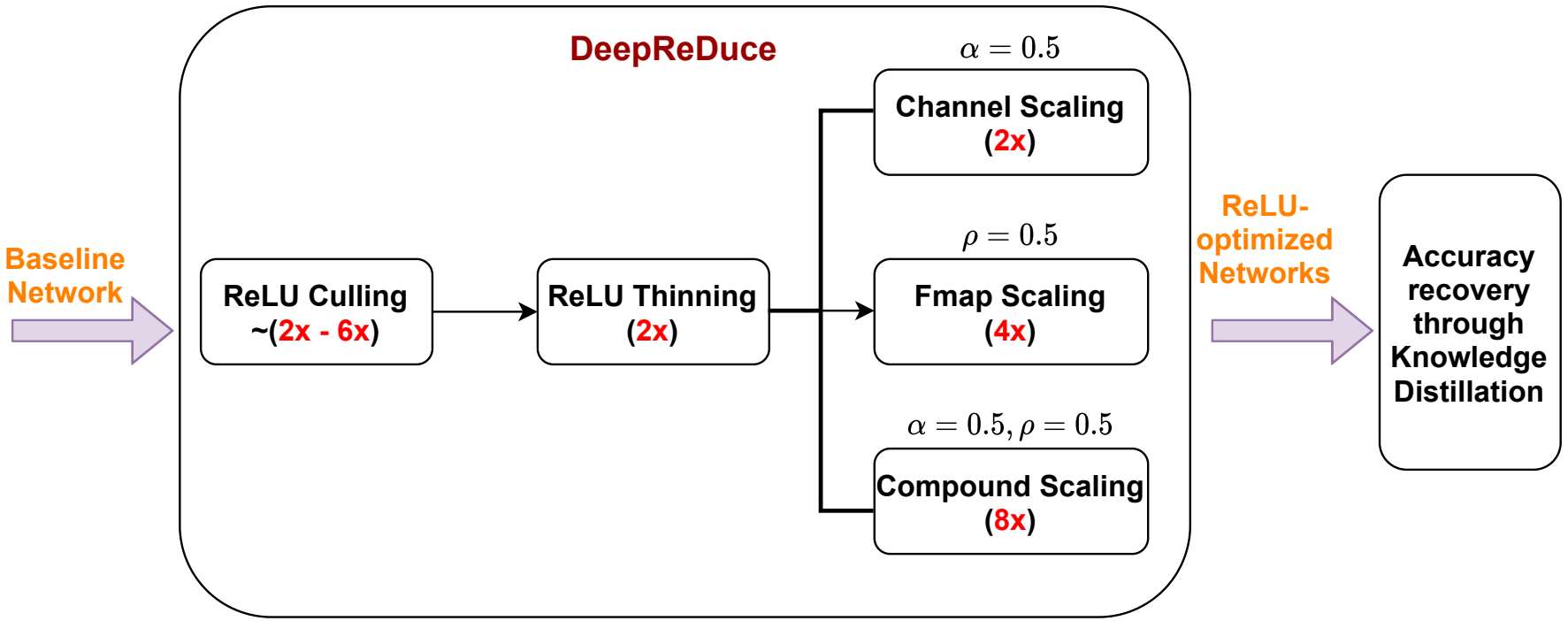
Models	Metrics	No ReLUs	Conv1	S1	S2	S3	S4
ResNet18	#ReLUs	0	66K	262K	131K	66K	33K
	W/o KD	18.49	46.2	61.9	67.63	67.4	58.9
	W/ KD	18.34	45.07	59.85	68.79	69.9	63.16
ResNet34	#ReLUs	0	66K	393K	262K	197K	49K
	W/o KD	18.16	45.42	60.77	69.47	70.04	57.44
	W/ KD	18.07	45.13	62.88	70.93	72.61	64.23

Observations

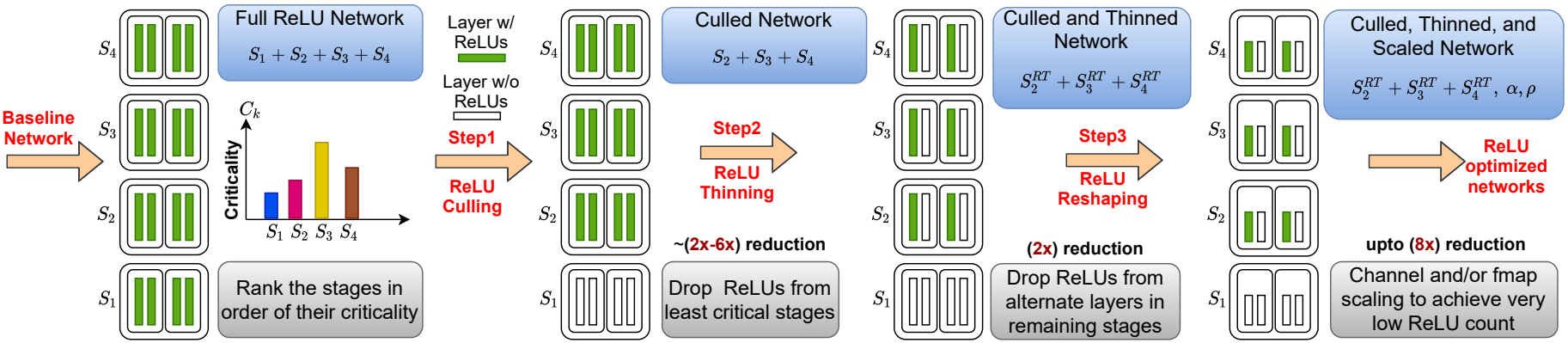
1. Higher # ReLUs *does not* guarantee higher accuracy.
2. ReLUs in some stages are *more critical* for accuracy than others.
3. ReLUs in deeper Stages/layers get *more benefit* from knowledge distillation than initial layers.

DeepReDuce: Big Picture

- **Reduce the huge search space**
  - Gradually drop the ReLUs from *coarse-grained* (Stage-wise) to *fine-grained* (layer-wise).
- **Exploiting the heterogeneity of ReLUs**
  - We drop (coarse to fine-grained) *following their criticality* to reduce the overall ReLU count with *minimal impact* on accuracy.
- **Synergistically apply Knowledge Distillation**
  - We account for *"gain from KD"* to decide the Stage-wise criticality of ReLUs.



DeepReDuce Method



Criticality Metric: For a network with D stages; $Acc[S_k]$ is accuracy of network with ReLUs dropped from entire network except the stage S_k.

$$C_k = \frac{(Acc[S_k]) - (\min_{i=1 \text{ to } D}\{Acc[Si]\})}{(\#ReLU[S_k])^{0.07}}$$

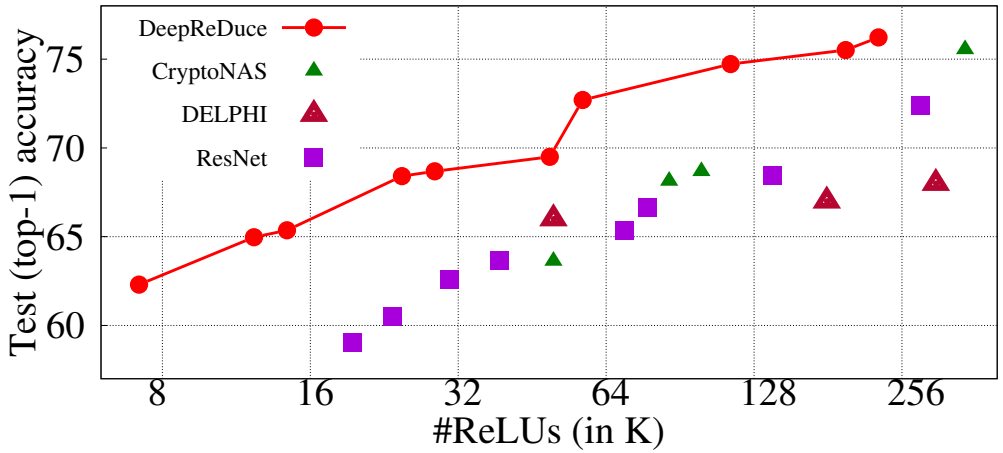
ReLU criticality on TinyImageNet: S_1 ReLUs are least and S_3 ReLUs are most critical.

Net	ResNet18				ResNet34			
	#ReLUs	W/o KD	W/ KD	C_k	#ReLUs	W/o KD	W/ KD	C_k
Baseline	2228K	61.28	-	-	3867K	63.06	-	-
S1	1049K	41.90	39.61	0.00	1573K	42.10	39.4	0.00
S2	524K	50.53	49.44	6.04	1049K	53.49	51.74	7.58
S2	262K	**51.93**	**54.34**	**9.50**	786K	**57.28**	**60.83**	**13.44**
S4	131K	46.89	51.46	8.02	197K	48.10	54.41	10.37

Results: Comparison with SOTA

Latency is in seconds.

	SOTA			DeepReDuce			Improvement		
	ReLUs	Acc.	Lat.	ReLUs	Acc	Lat	ReLU	Acc.	Lat.
CryptoNAS	344K	75.5	7.50	197K	75.50	3.94	1.75×	0.0	1.9×
	100K	68.7	2.30	28.6K	68.70	0.738	3.5×	0.0	3.1×
	86K	68.1	2.00	28.6K	68.70	0.738	3×	0.6	2.7×
	50K	63.6	1.67	**12.3K**	**65.00**	**0.455**	4×	1.4	3.7×
DELPHI	300K	68	6.5	28.7K	68.70	0.738	10.5×	0.7	8.8×
	180K	67	4.44	24.6K	68.41	0.579	7.3×	1.4	7.7×
	50K	66	1.23	**49.2K**	**69.50**	**1.19**	1×	3.5	1×



Comparison on CIFAR-100 datasets

- **Improvement over SOTA**
  - 3.5% accuracy improvement at iso-ReLU
  - 3.5× ReLU saving at iso-accuracy
- **Comparison with ResNet**
  - ResNet's accuracy drops rapidly at lower ReLU counts
  - Accuracy improvement increases at lower ReLU counts

Analysis of Pareto Points

Accuracy on CIFAR-100 and latency measured in seconds.

Culled	Thinned	Reshaped Ch.	Reshaped Fmap	#ReLUs	Acc.(%)	Lat.	Acc / ReLU
S_1	NA	NA	NA	229.38K	76.22	4.61	0.332
S_1+S_4	NA	NA	NA	196.61K	75.51	3.94	0.384
S_1	$S_2+S_3+S_4$	NA	NA	114.69K	74.72	2.38	0.651
S_1	$S_2+S_3+S_4$	0.5×	NA	57.34K	72.68	1.37	1.27
S_1+S_4	S_2+S_3	0.5×	NA	49.15K	69.50	1.19	1.45
S_1	$S_2+S_3+S_4$	NA	0.5×	28.67K	68.68	0.74	2.40
S_1+S_4	S_2+S_3	0.5×	NA	24.57K	68.41	0.56	2.78
S_1	$S_2+S_3+S_4$	0.5×	0.5×	14.33K	65.36	0.52	4.56
S_1+S_4	S_2+S_3	0.5×	0.5×	12.28K	**64.97**	**0.45**	5.29
S_1	$S_2^\star+S_3^\star+S_4^\star$	0.5×	0.5×	7.17K	**62.30**	**0.21**	8.69

- **Takeaway 1:** ReLU saving is (proportionately) translated into latency saving. - We achieve 65% accuracy with 450mS latency
- **Takeaway 2:** Accuracy Per ReLU keeps increasing as ReLU count decreases - Least accurate model is 13.9% less accurate, and it has 32× fewer ReLUs with 26.2× higher Acc/ReLU.

DeepReDuce Models vs. Shallow ResNets

Iso-ReLU and **Iso-conv** layers comparison on CIFAR-100. Consecutive linear layers (withing a stage) is *merged* to reduce the #conv layers.

Network	#Conv	#ReLUs	W/o KD	W/ KD
ResNet10, α=0.5	9	155.6K	71.3	72.5
$S_2^{RT} + S_3 + S_4^{RT}$	17	147.6K	71.7	74.8
ResNet10, ρ=0.5	9	47.1K	64.7	68.1
$S_2^{RT} + S_3^{RT}$	11	49.2K	67.8	71.0
ResNet9, α=0.5, ρ=0.5	8	30.7K	62.6	66.2
$S_2^{RT} + S_3^{RT} + S_4^{RT}$, ρ=0.5	8	28.7K	64.4	68.5
$S_2^{RT} + S_3^{RT}$, α=0.5	7	24.6K	66.0	68.1

- **DeepReDuce models outperform shallower models**

  -DeepReDuce models at iso-ReLU and iso-conv layers are more accurate (w/ & w/o KD)

  -DeepReDuce and ResNet9 both have eight convolution layers; however, the former uses 2K fewer ReLUs and is 2.3% more accurate

Comparison with SOTA Channel Pruning Method

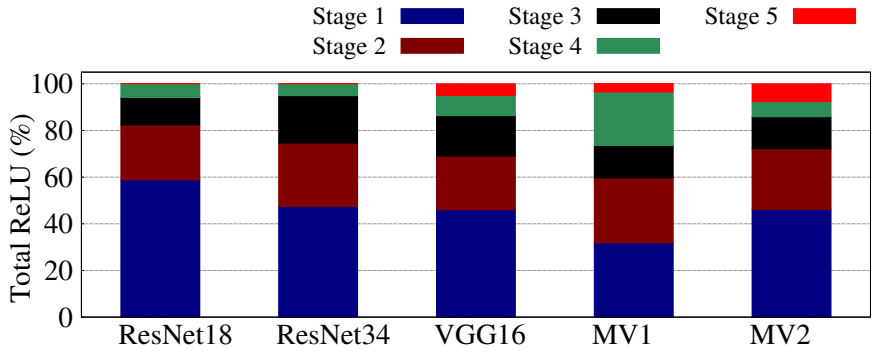
Comparison with SOTA channel pruning [He, CVPR'20] on ResNet56 with CIFAR-10/100.

	Method	Baseline Acc.(%)	Pruned Acc.(%)	Acc. ↓(%)	FLOPs	ReLUs
C10	Channel pruning	93.59	93.34	-0.25	59.1M	311.7K
	DeepReDuce	93.48	94.07	+0.59	87.7M	221.2K
			93.16	**-0.32**	**66.5M**	**147.5K**
C100	Channel pruning	71.41	70.83	-0.58	60.8M	311.7K
	DeepReDuce	70.93	73.66	+2.57	87.7M	221.2K
			71.68	**+0.59**	**66.5M**	**147.5K**

- ResNet56 has three stages and their criticality order is $S_3 > S_2 > S_1$.
- For above bold DeepReDuce models, we culled the S_1 stage, thinned S_2, and combine consecutive linear layers to reduce FLOPs.
- At comparable FLOPs and accuracy DeepReDuce models provide 2× more ReLU saving.

Generality of DeepReDrop

ReLU count in initial stages are *higher* than deeper stages in all the DNNs shown in the diagram.



MobileNetV1 on CIFAR-100

Stages	#ReLUs	W/o KD	W/ KD	C_k
S_1	131K	33.06	34.16	0.00
S_2	28.7K	49.64	50.65	12.55
S_3	14.3K	55.56	54.20	15.13
S_4	23.5K	**57.37**	**61.10**	**20.29**
S_5	3.6K	42.32	45.45	8.69

MobileNetV2 on CIFAR-100

Stages	#ReLUs	W/o KD	W/ KD	C_k
S1	196.6K	37.82	34.25	0.00
S2	110.6K	49.83	46.93	9.12
S3	58.4K	54.74	53.06	14.15
S4	27.6K	**57.08**	**57.28**	**18.26**
S5	32.4K	48.42	50.49	12.73

Similar to ResNet18 and ResNet34 on CIFAR-100 (& TinyImageNet), ReLUs in initial stage are *least critical* and that in penultimate stage are *most critical*.

Conclusion and Future Work

- ReLUs in DNNs exhibit heterogeneity which cab be exploited to efficiently and gradually reduce the ReLU count.
- Consecutive linear layers (within a stage) can be combined to reduce the FLOPs with negligible impact on accuracy.
- ReLU-aware DNN's architectures are the **need of the day** since FLOPs reduction mechanism *does not ensure* the ReLU Reduction.

  -Group convolution and Depthwise convolution reduces the FLOPs and parameters; *however*, #ReLUs remains same.

  - Slimed networks trade FLOPs and parameters reduction with *increased* ReLU count

Networks	#Params	#FLOPs	#ReLUs
MobileNetV1	3.29M	46.45M	**411.65K**
MobileNetV2	1.66M	15.90M	**425.60K**

  - The (slimed) MobileNetV2 has 3× fewer FLOPs and 2× fewer parameters than MobileNetV1; however, the former has *higher ReLU count* than the later.

Thanks for your attention..!!!

Thanks for your attention..!!!
Q & A?