Optimization and Deployment Analysis of Split-Attention Network (ResNeSt)

Team 22

Presented By Haodong Ge & Xirui Fu

Executive Summary:

- Objective 1: Implement and optimize 50-Layer Split-Attention Networks (ResNest-50) , which is developed from ResNet.
    - Innovation point: implement split-attention technique on basic block structure, and compare it with bottle-neck structure from original ResNeSt paper.
    - (Basic Block vs Bottleneck)
- Objective 2: Deploy the target model on the cloud platform & mobile framework . Compare the performance in metrics including test accuracy, model size and inference time.
    - (Pytorch vs Caffe2)

Problem Motivation:

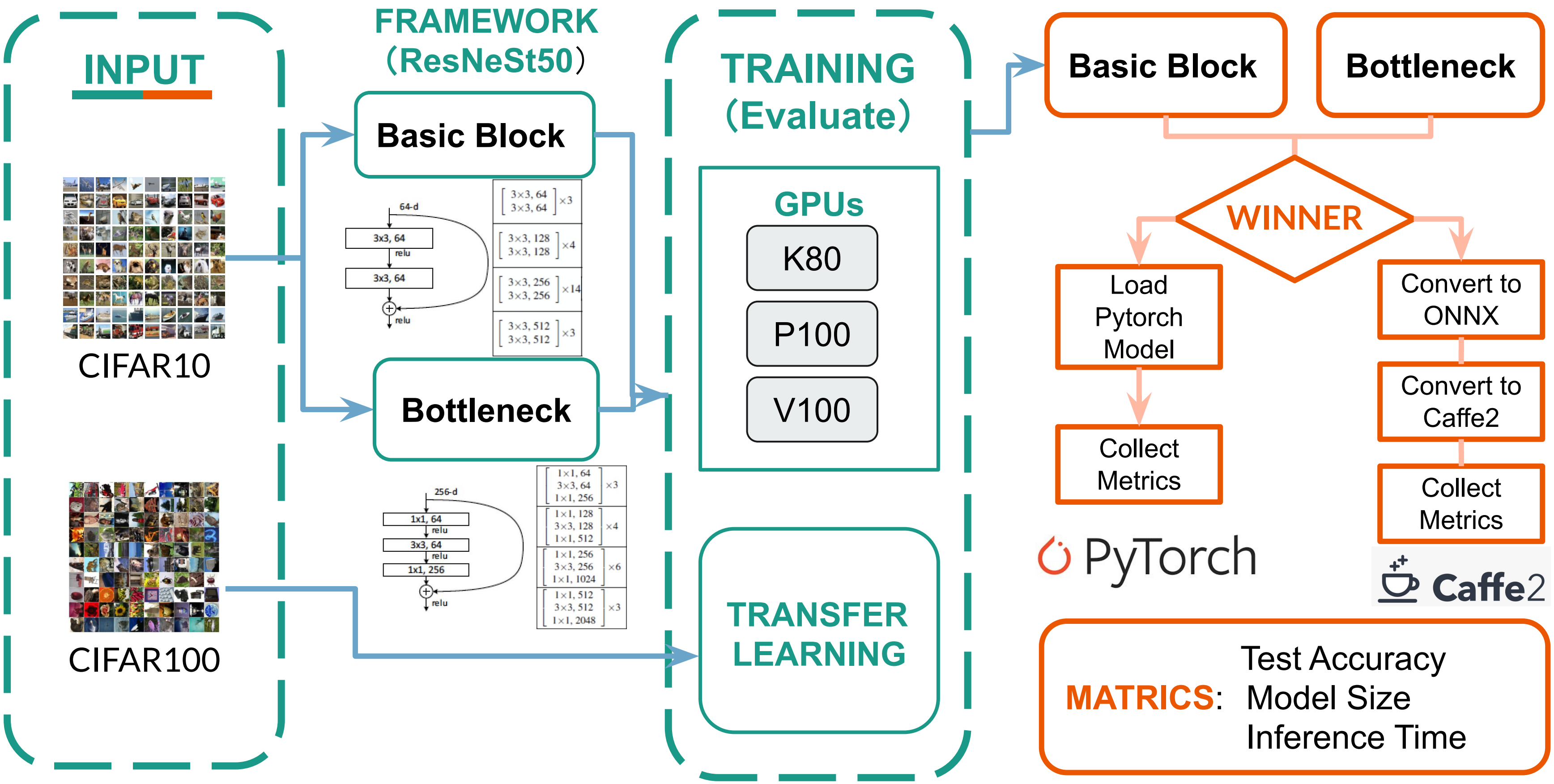
Model Framework

- ResNeSt maintains the overall ResNet structure and outperform other networks with similar model complexity.
- The tradeoff between basic block and bottleneck structure changes along with depth of the framework. The tradeoff's effect is unclear at 50 Layers.

Deployment

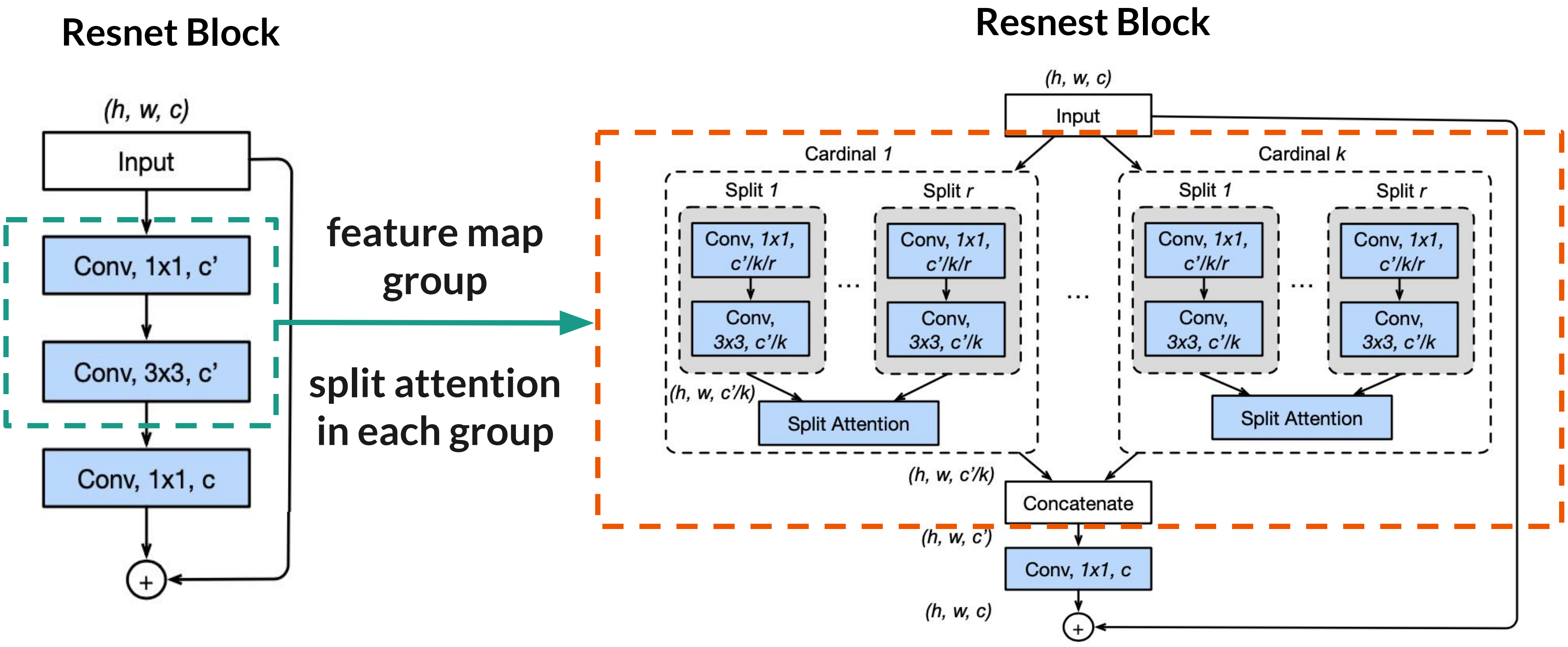
- The tradeoff between mobile DL framework and traditional ones has been popular topic in recent years
  - (Lighter & Faster vs Stronger)
- The comparison between two frameworks is meaningful for integrated deployment in industry and cross-platform DL ecosystem.

Solution Diagram

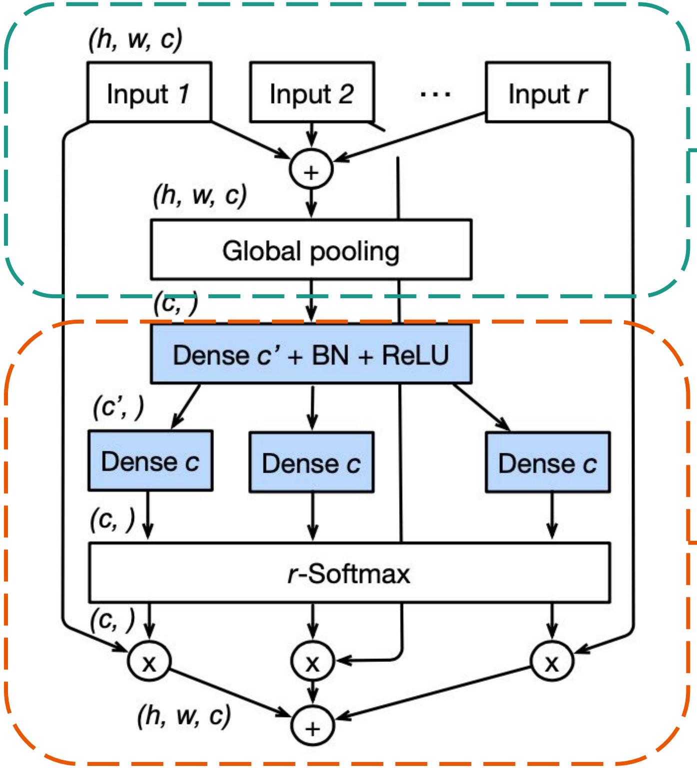


ResNeSt Framework

Background Work:



Background Work: Split Attention



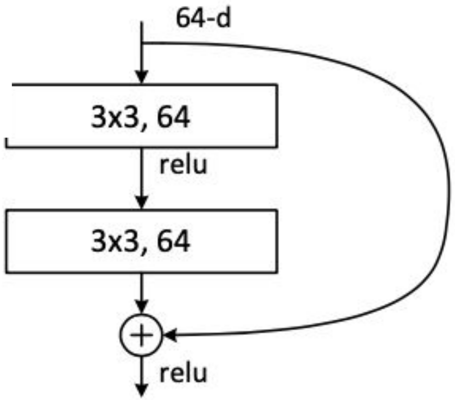
Global pooling:

Gather global contextual information with **embedded channel-wise statistics** across spatial dimensions.

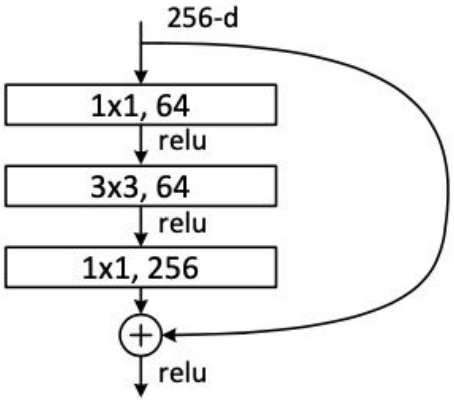
Channel-wise attention

Each featuremap channel is **a weighted combination of splits** in a cardinal group

Background Work: Basic Block VS Bottleneck



Basic Block



Bottleneck

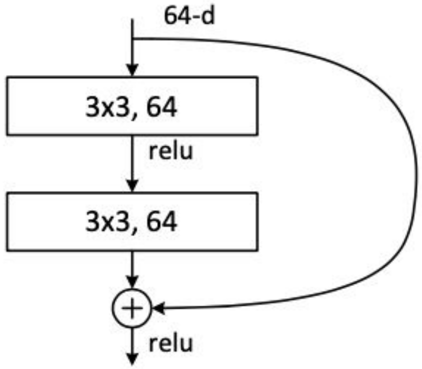
- **Why use bottleneck?**

  The bottleneck architecture is used in **very deep networks** for **computational efficiency**

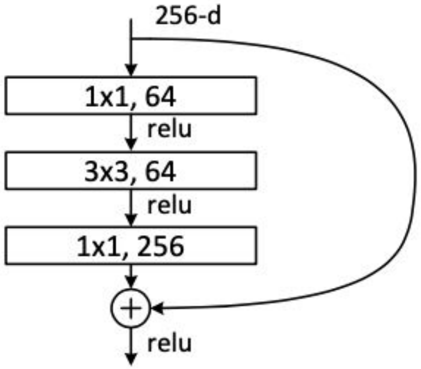
- **How to reduce computation**

  1x1 convolution filter can be used to **change the dimension (either increase or decrease)**

Technical Challenges: Accuracy OR Computational Efficiency



Basic Block

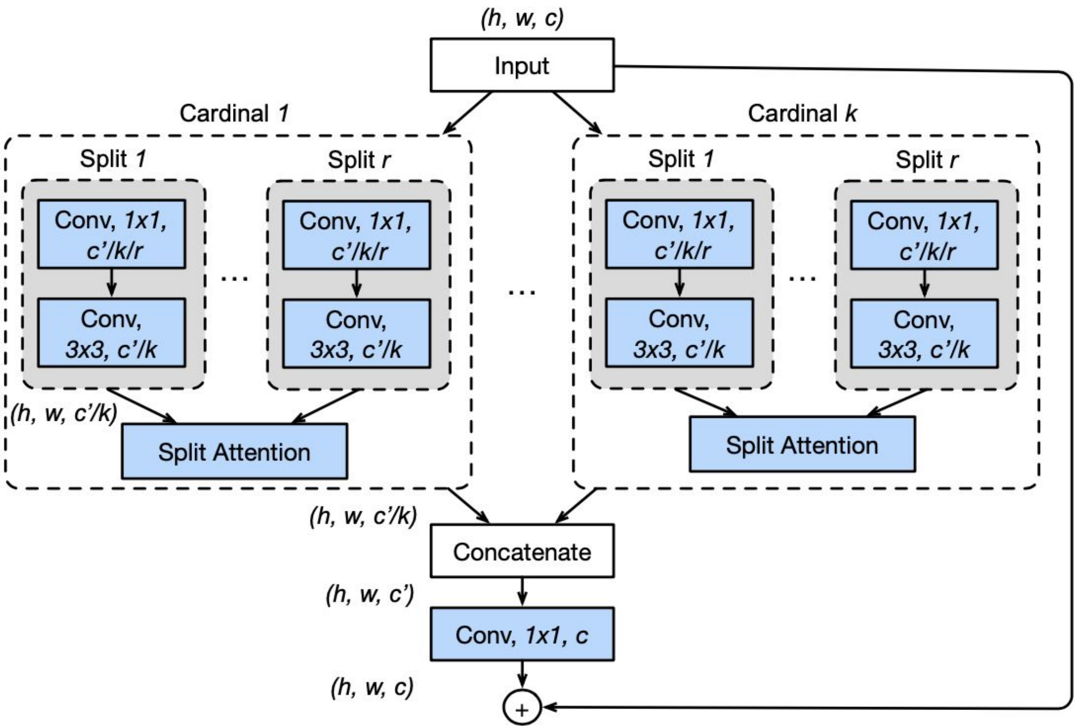


Bottleneck

- The tradeoff between basic block and bottleneck structure changes along with the depth of the model.
- However, the tradeoff is unclear at 50 layers.

Technical Challenges: How to apply split attention to basic block

Bottleneck



Approach:

Basic block and bottleneck structure:

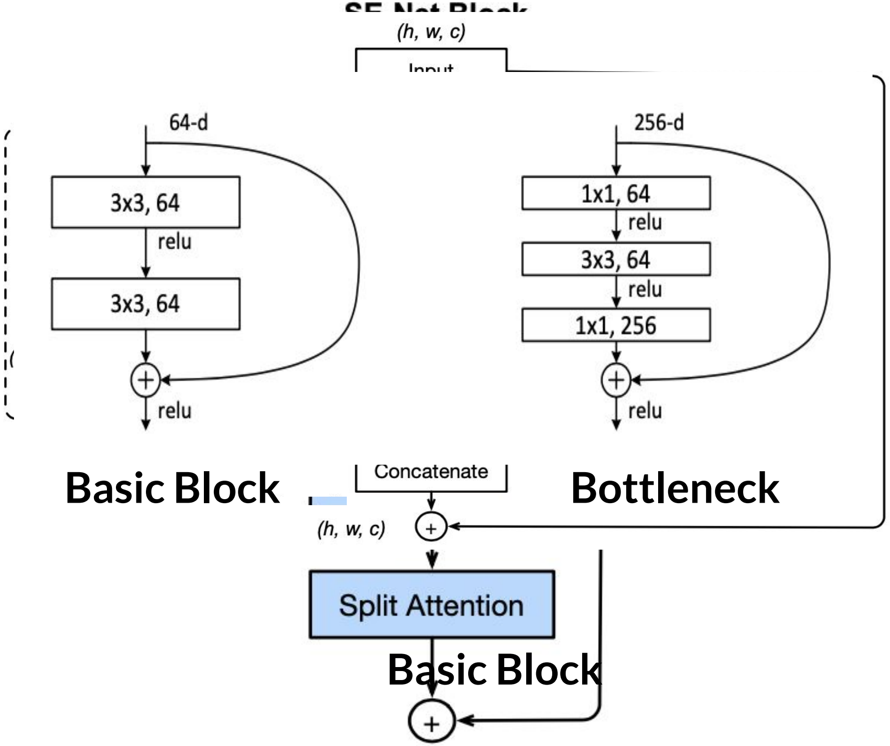
- Basic block: 3x3 conv layer, 3x3 conv layer
- Bottleneck : 1x1 conv layer, 3x3 conv layer, 1x1 conv layer

First 1x1 conv layer: <u>reduce dimension for computation</u>

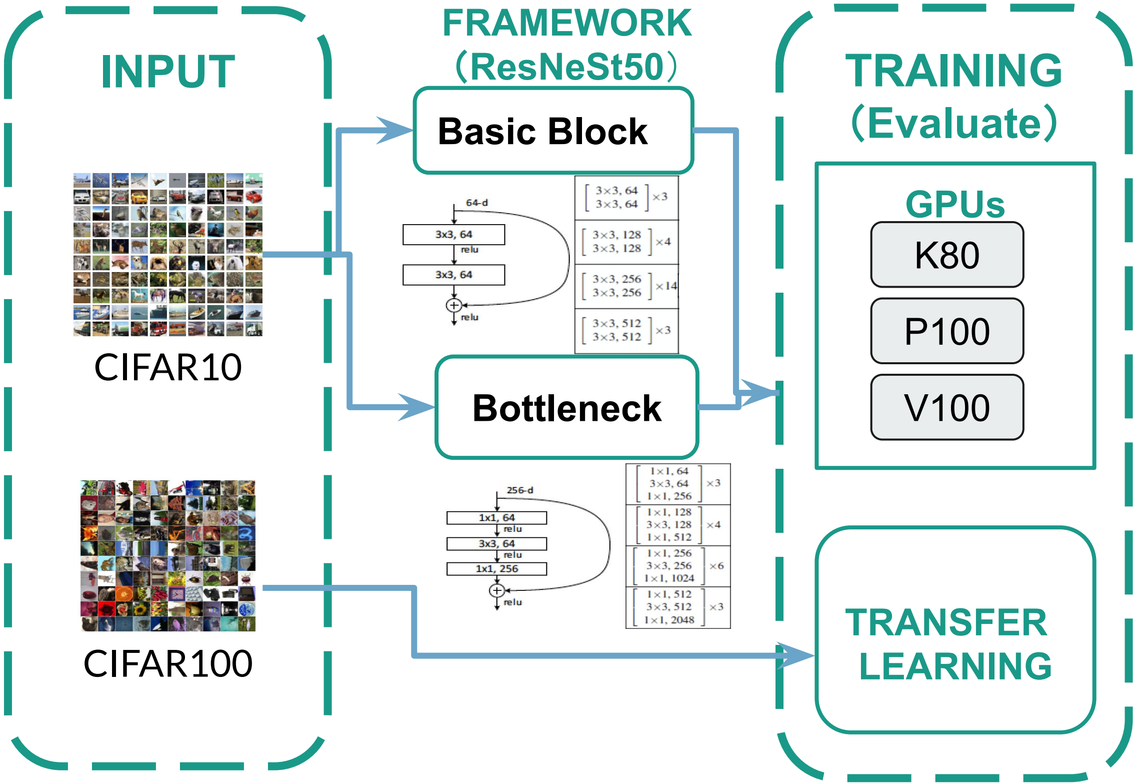
Second 1x1 conv layer: <u>increase dimension for shortcut</u>

From **SE-Net**, the output of split attention can be direct input for the shortcut.

Therefore, we can drop the third layer and adjust the filter size the same as the input dimension.



Solution diagram



Basic block	Bottleneck
$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times14$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$
$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$

Implementation details

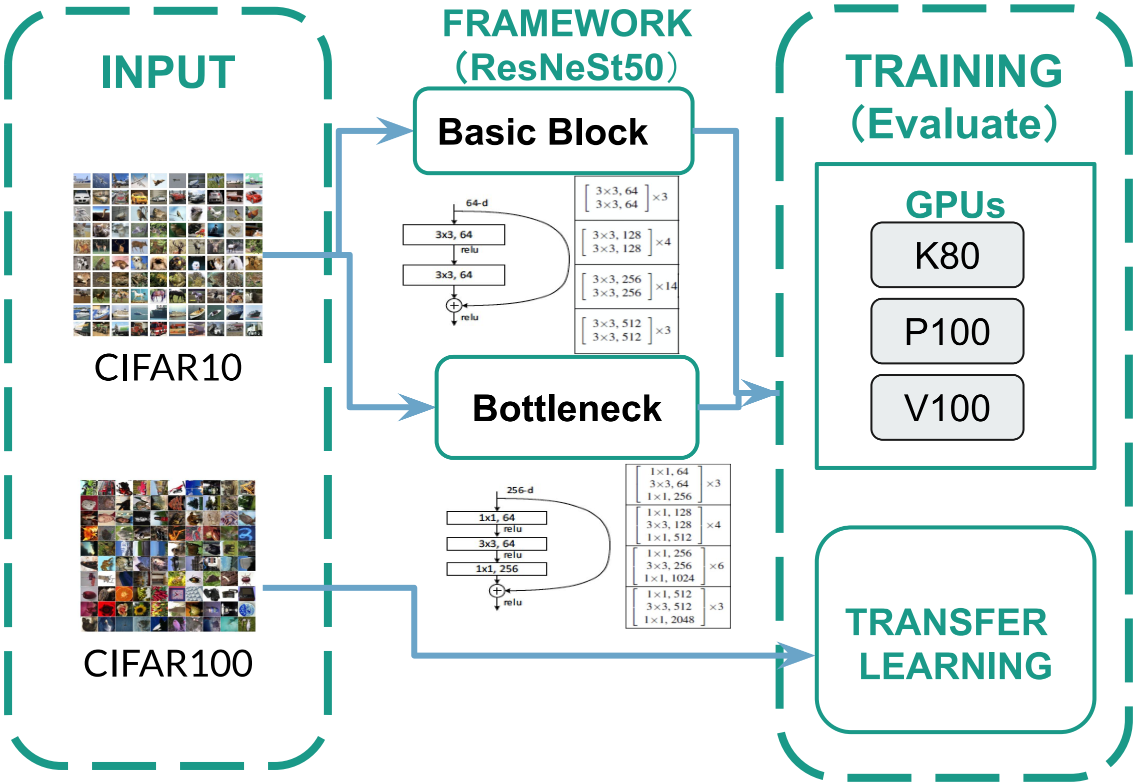
Platform: GCP
Framework: Pytorch

Dataset: CIFAR10

- Learning rate: 0.1
- Lr scheduler: gamma=0.1, milestone=[80,210]
- Criterion: crossEntropy
- Optimizer: SGD(weight_decay=0.0001,momentum=0.9)
- Epoch: 250
- Batch size: 128

Dataset: CIFAR100

- Learning rate: 0.001
- Lr scheduler: gamma=0.1, step size=60
- Criterion: crossEntropy
- Optimizer: SGD(momentum=0.9)
- Epoch: 200
- Batch size: 64

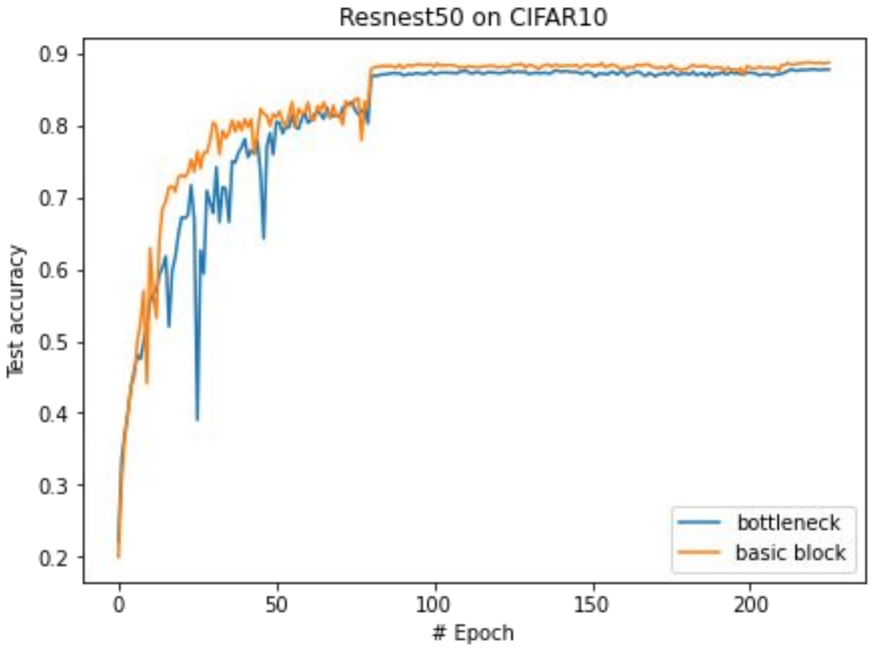


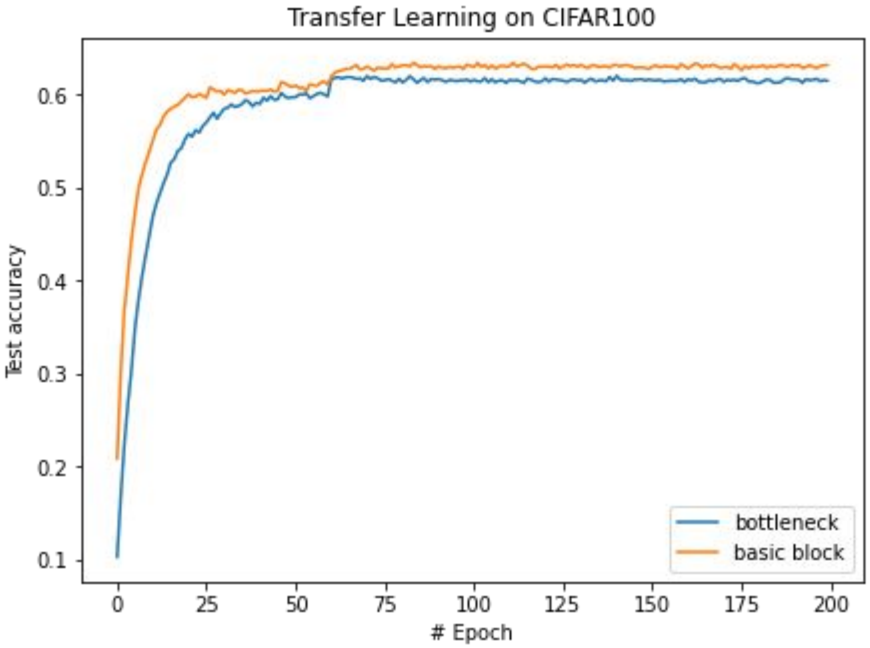
Experiment evaluation

Accuracy					
	K80	**P100**	**V100**	**Average**	**Transfer learning**
Basic block	0.8862	0.8923	0.8907	0.8897	0.6323
Bottleneck	0.8705	0.8742	0.8765	0.8737	0.6152
Training Time (s / epoch)					
Basic block	302	36	25.5		
Bottleneck	149	31	22		

Tradeoff - Basic block: more accurate; Bottleneck: more computational efficient

Experiment evaluation





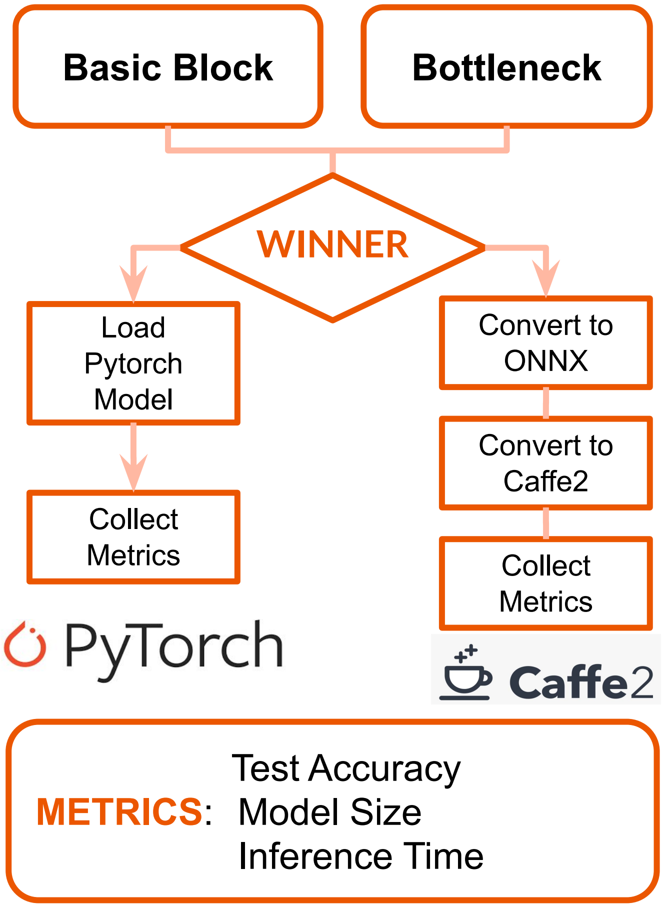
Basic block has higher accuracy and converges fast.

DEPLOYMENT ANALYSIS

Deployment Implementation

- **Pytorch** model is loaded on **GCP** (Google Cloud Platform) for inference on single image and two test sets

- To get the **Caffe2** Model
  - Export the pretrained model to ONNX model
  - Convert **ONNX** to Caffe2 model
  - Simulate Caffe2 inference with ONNX model on Caffe2 backend



Inference on single image

As the table shown on the right, scores of the two models are super close with difference smaller than **10^-5**

Predict #	Predict Class	.pt Model Score	.onnx Model Score
1	Ship	**0.994803439**	**0.99480238**
2	Horse	**0.002122742**	**0.002120206**
3	Dog	**0.001315854**	**0.00131847**
4	Truck	**0.0011049229**	**0.0011057975**
5	Frog	**0.000377598**	**0.000377779**
6	Bird	**0.000216783**	**0.000216705**
7	Airplane	**4.08936e-05**	**4.08939e-05**
8	Deer	**1.4758e-05**	**1.4758e-05**
9	Automobile	**2.767e-06**	**2.7671e-06**
10	Cat	**2.422e-07**	**2.422e-07**

Inference on testsets

- There are two test sets with different sizes
  - The large test set is exactly CIFAR10 test set with 10k images
  - The small test set is manually labeled (not in CIFAR) with 50 images
- The avg_ time indicates mean value of all images' inference time in test set.

	Test_acc_s (%)	Avg_time_s (second)	Test_acc_l (%)	Avg_time_l (second)	Model_size (mb)
Pytorch	0.9	0.05145	0.8923	0.05242	128
Caffe2	0.88	0.02854	0.8812	0.02878	115

Caffer2: lighter and faster; Pytorch: slightly better accuracy

Conclusion:

Model Framework

- Basic block is more accurate and converges faster, while bottleneck takes less time to train.
- There is a tradeoff between accuracy and computational efficiency.

Deployment

- Compared to the traditional DL framework like pytorch, Mobile DL framework like Caffe2 still has advantage on model size and significantly shorter inference time.
- However, the model performance will be slightly compromised after compressing.

Limitation & Future Work

- **Model:**
  - Tune the hyperparameters of Resnest50, e.g. cardinality and radix.
  - Various datasets should be used in model selection and transfer learning for better evaluation, like ImageNet.
  - Achieve object detection.
- **Deployment:**
  - Due to the limited time, only caffe2 model is practiced in this project with simulation on ONNX model on caffe2 backend.
  - More mobile DL frameworks should be explored like CoreML, TFlite, etc.
  - Models should be implemented on different hardware platforms like Iphone (IOS) and Samsung (Andriod).

Thank You

https://github.com/xirui-geverson-dl/optimization-and-deployment-of-ResNeSt

Question, Comments and Concerns?