Graph-Based Global Reasoning Networks, CVPR (2019)

Gee Ho Kim

Seoul National University

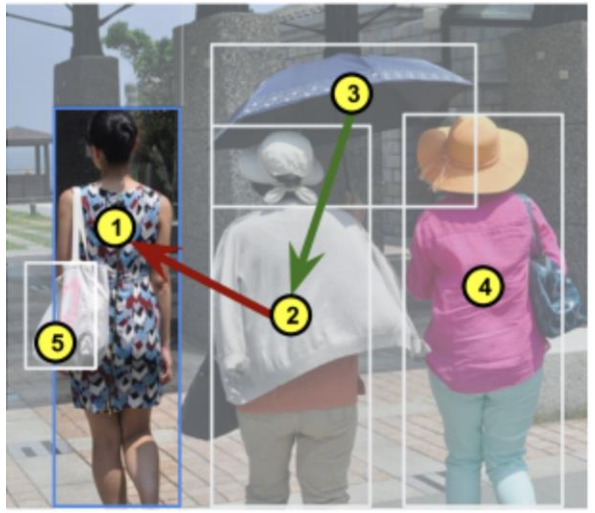
Introduction

- Relational reasoning between distant regions in coordinate space

(a) GT: Playing TV Game

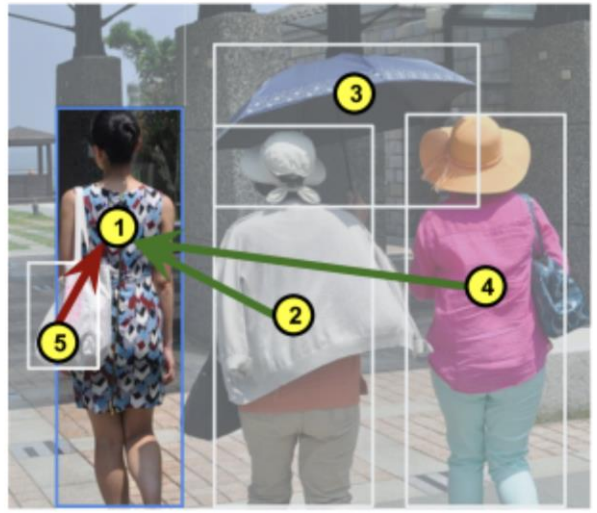
Question: Is there a person to the left of the woman holding a blue umbrella?

Answer: Yes



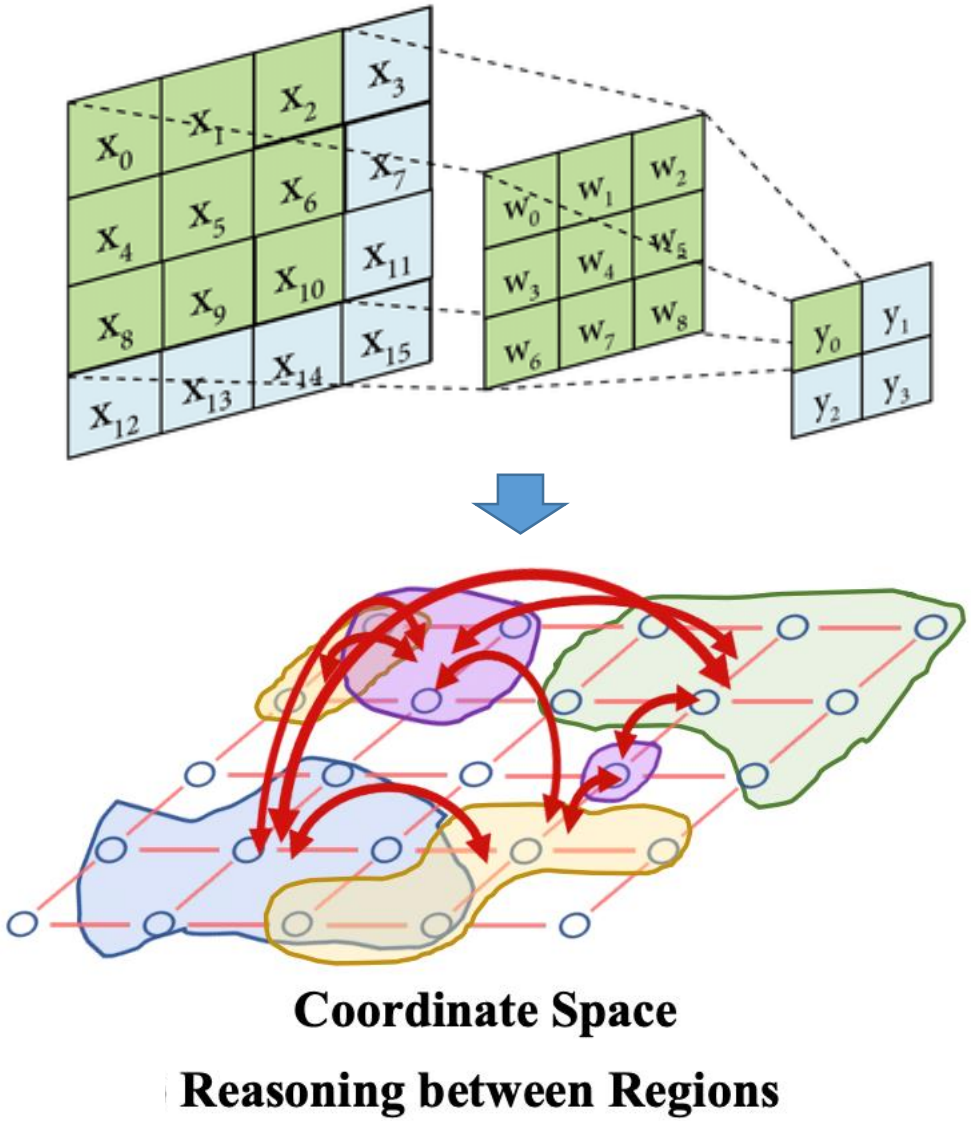
Question: Is the left-most person holding a red bag?

Answer: No



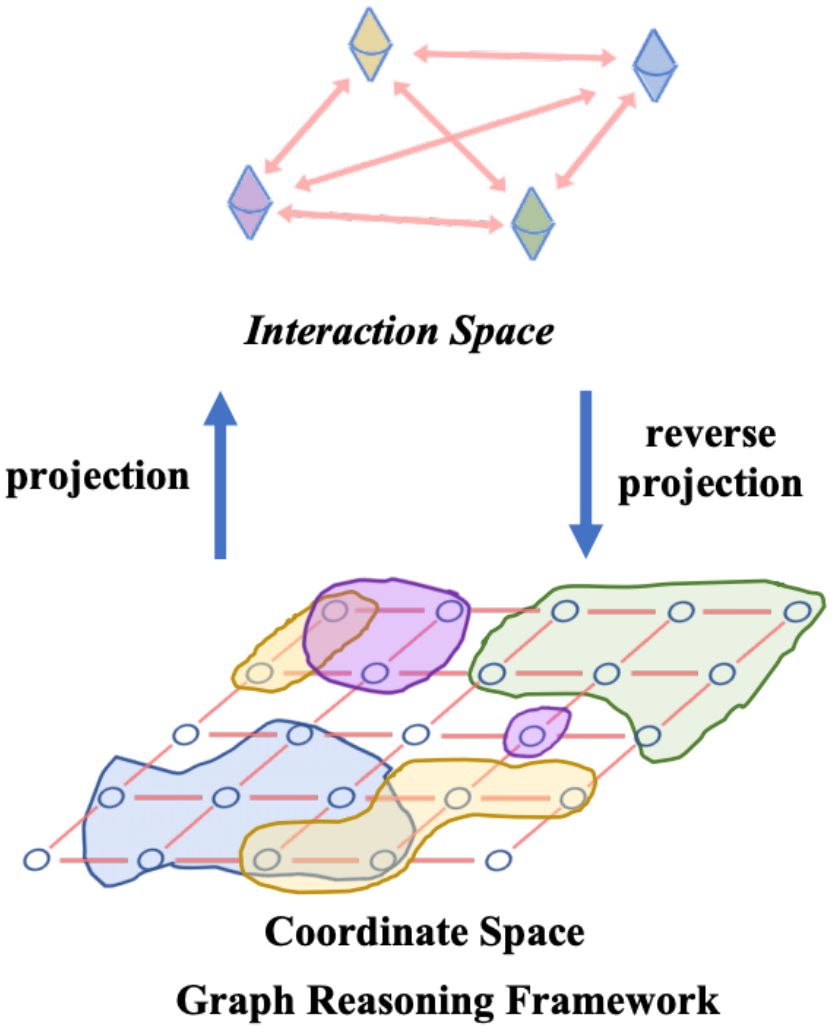
Introduction

- In classical CNNs, a stack of the convolutional layers for sufficient receptive field.
- The paper propose a global reasoning unit (GloRe) which enables interaction between distant parts of the input data in early stages of CNN model.

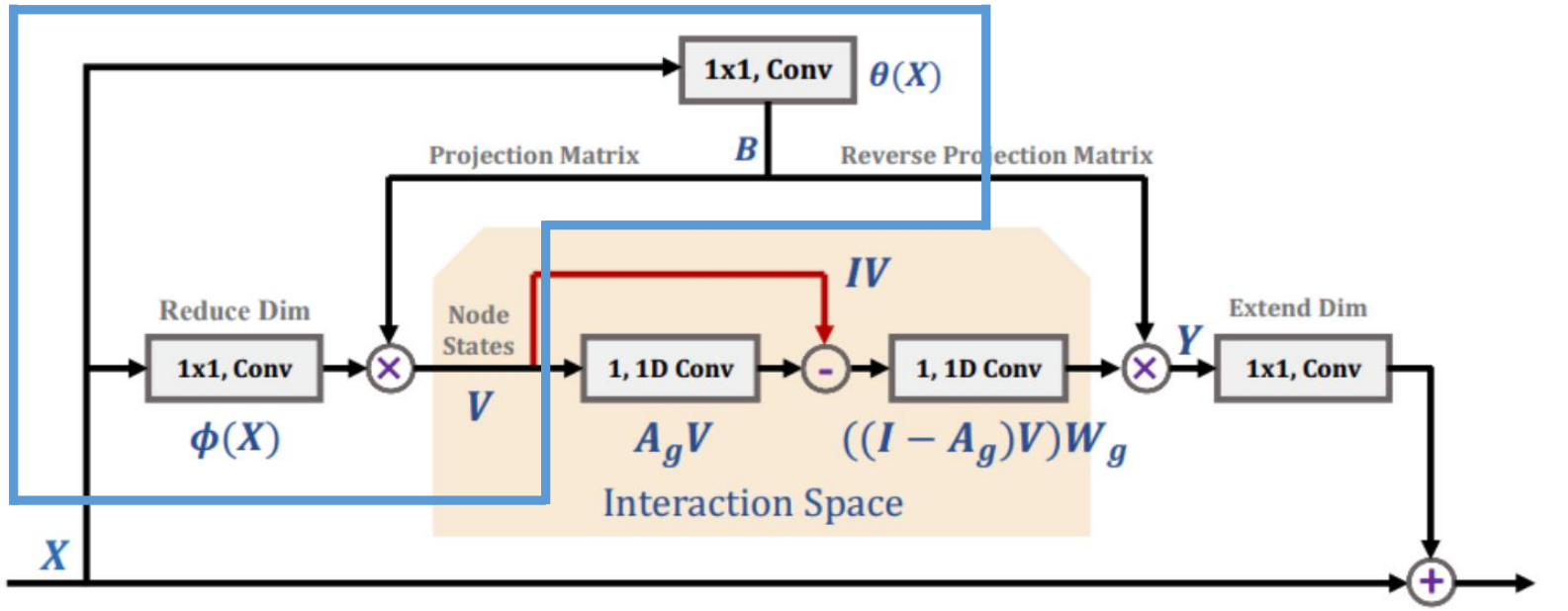


Relation reasoning

- Global aggregation from a set of features
- Projection to an interaction space where relational reasoning is computed
- Reverse projection to coordinate space for down-stream tasks

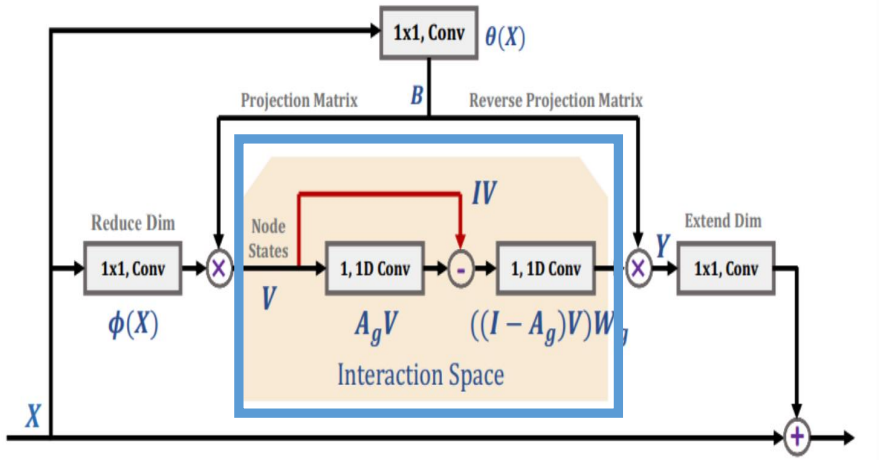


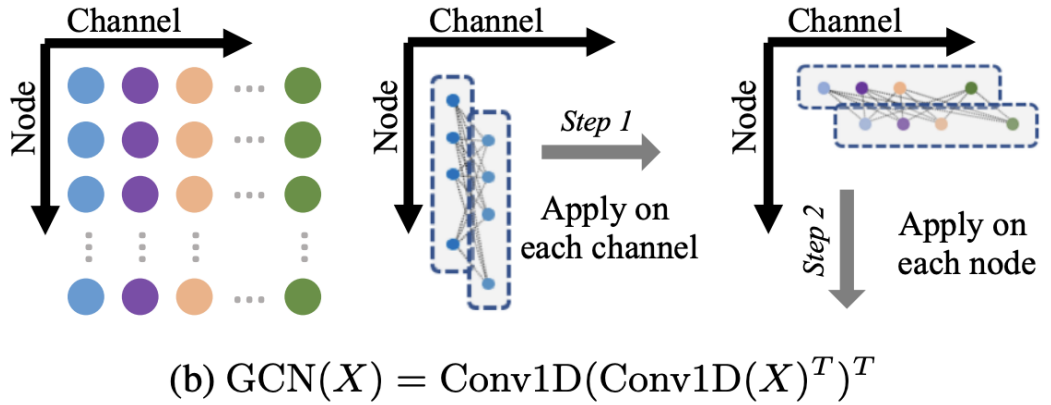
Global Reasoning (GloRe) Unit



- GloRe unit projects features over coordinate space into interaction space (graph) by project function ($\phi(X) \in \mathbb{R}^{N \times C}, V \in \mathbb{R}^{L \times C}, B \in \mathbb{R}^{N \times L}$)
- The projection function is formulated as a linear combination of original features with learnable projection matrix (B)
- Node states (V) is obtained by global weighted-average pooling with B

GloRe Unit

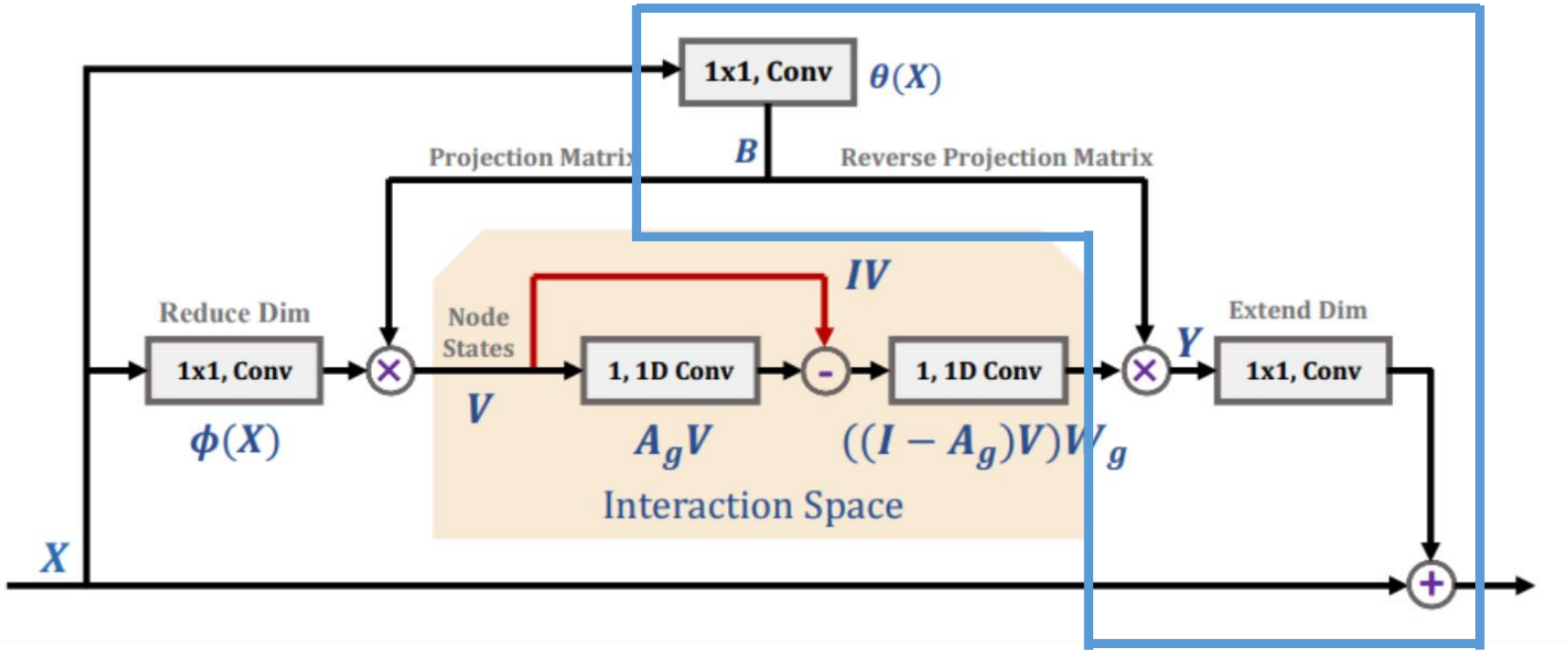




(b) $\mathrm{GCN}(X) = \mathrm{Conv1D}(\mathrm{Conv1D}(X)^T)^T$

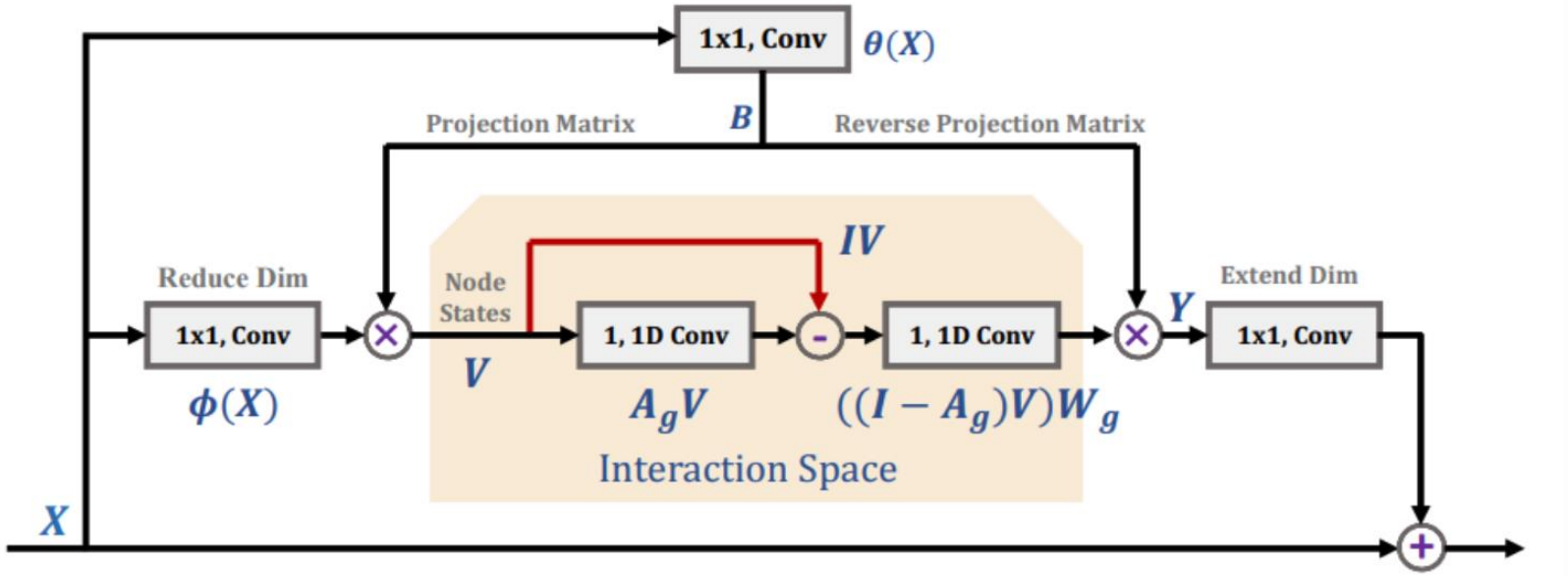
- Treat V as nodes of a fully connected graph
- Reasoning on the graph by learning edge weights via graph convolution.
- Two-direction 1D convolution for implement graph convolution.

GloRe Unit



- Reverse project from interaction space to coordinate space
- Revere Projection matrix $D = B^{\mathrm{T}}$
- Another convolution layer for migration of the information back to original space forming a residual path

GloRe Unit



- Simple global reasoning method by global weighted-average pooling and GCN
- Compatibility and Light-weight
  - Residual nature
  - Complementary to both shallow and deeper networks

Results

- Image classification on ImageNet

Method		Res3	Res4	GFLOPs	#Params	Top-1
ResNet50 [16]	Baseline			4.0	25.6M	76.2%
	GloRe (Ours)		+3	5.2	30.5M	**78.4%**
	GloRe (Ours)	+2	+3	6.0	31.4M	78.2%
SE-ResNet50 [18]	Baseline			4.0	28.1M	77.2%
	GloRe (Ours)		+3	5.2	33.0M	**78.7%**
ResNet200 [16]	Baseline			15.0	64.6M	78.3%
	GloRe (Ours)		+3	16.2	69.7M	79.4%
	GloRe (Ours)	+2	+3	16.9	70.6M	**79.7%**
ResNeXt101 [34] (32 × 4)	Baseline			8.0	44.3M	78.8%
	GloRe (Ours)	+2	+3	9.9	50.3M	**79.8%**
DPN-98 [9]	Baseline			11.7	61.7M	79.8%
	GloRe (Ours)	+2	+3	13.6	67.7M	**80.2%**
DPN-131 [9]	Baseline			16.0	79.5M	80.1%
	GloRe (Ours)	+2	+3	17.9	85.5M	**80.3%**

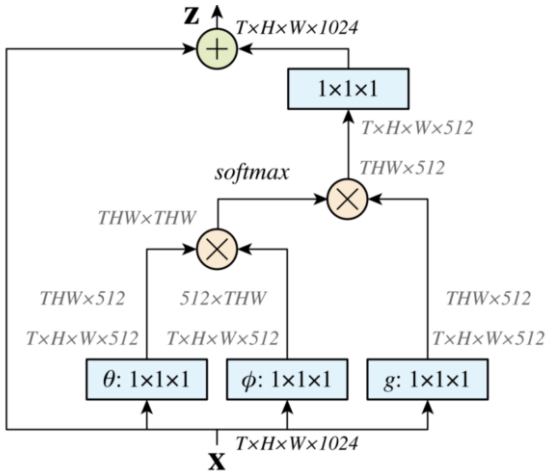
Results

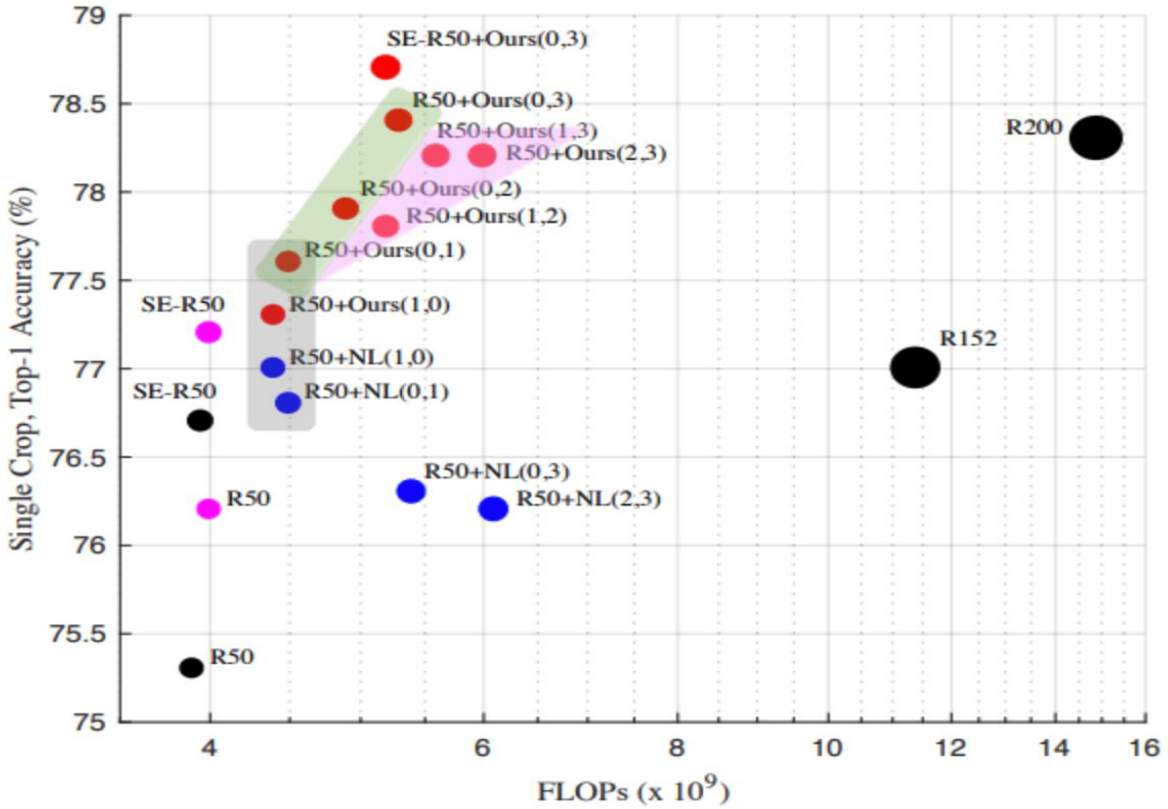
- Image segmentation on Cityscapes

Method	Backbone	IoU cla.	iIoU cla.	IoU cat.	iIoU cat.
DeepLab-v2 [4]	ResNet101	70.4%	42.6%	86.4%	67.7%
PSPNet [36]	ResNet101	78.4%	56.7%	90.6%	78.6%
PSANet [37]	ResNet101	80.1%			
DenseASPP [35]	ResNet101	80.6%			
FCN + 1 GloRe unit	ResNet50	79.5%	60.3%	91.3%	81.5%
FCN + 1 GloRe unit	ResNet101	**80.9%**	**62.2%**	**91.5%**	**82.1%**

Results

- Image classification on ImageNet
- Comparison with Non-local block [1]
  - Non-local block
    - Deliver long-range information by pixel-wise self-attention
  - $N = \frac{1}{4} C$





[1] Wang et al., **Non-local neural networks,** In Computer Vision and Pattern Recognition (CVPR), 2018.

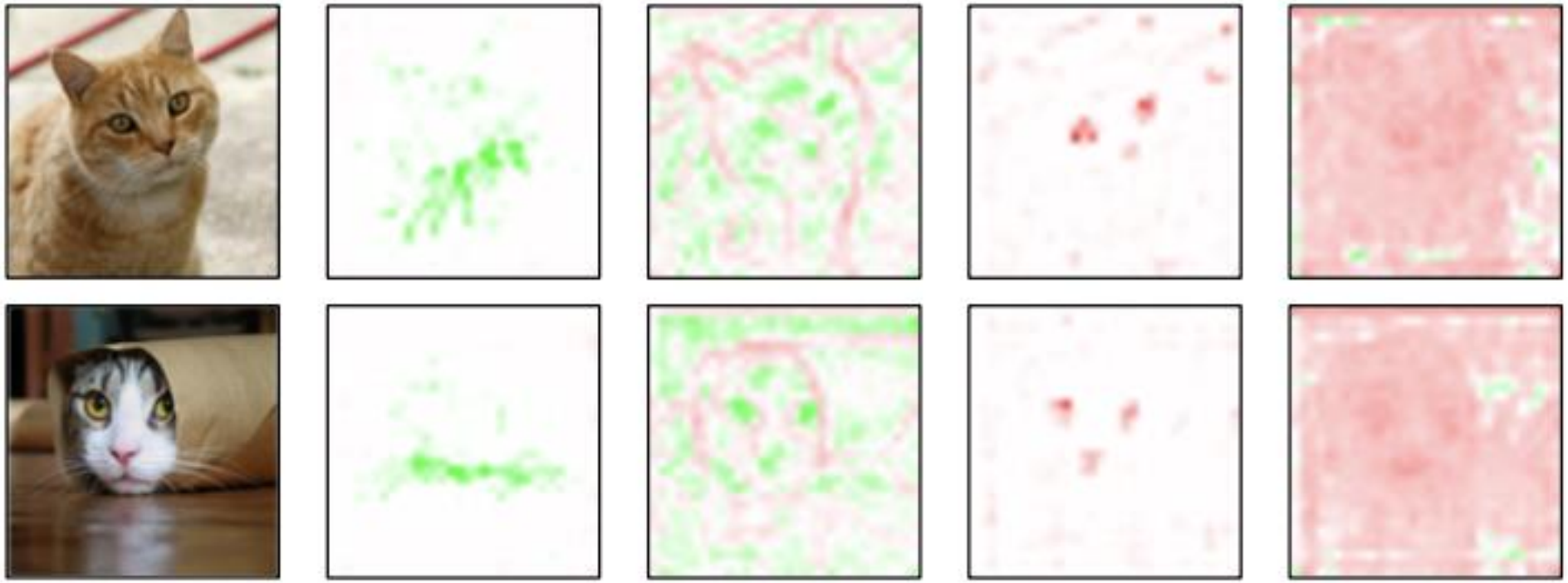
Results

- Video action recognition on Kinetics-400

Method	Backbone	Frames	FLOPs	Clip Top-1	Video Top-1
I3D-RGB [2]	Inception-v1	64	107.9 G	–	71.1%
R(2+1)D-RGB [29]	ResNet-xx	32	152.4 G	–	72.0%
MF-Net [8]	MF-Net	16	11.1 G	–	72.8%
S3D-G [34]	Inception-v1	64	71.4 G	–	74.7%
NL-Nets [31]	ResNet-50	8	30.5 G	67.12%	74.57%
GloRe (Ours)	ResNet-50	8	28.9 G	68.02%	**75.12%**
NL-Nets [31]	ResNet-101	8	56.1 G	68.48%	75.69 %
GloRe (Ours)	ResNet-101	8	54.5 G	68.78%	**76.09%**

Results

- Visualization of the learned projection weights



Conclusion

- A novel global reasoning approach
  - Projection of globally aggregated features over coordinate space into an interaction space
  - Relation reasoning in the interaction space
  - Distributed back to coordinate space
- Global Reasoning unit (GloRe unit)
- Performance boost for a wide range of backbones and various tasks
- Code Link: https://github.com/facebookresearch/GloRe.git
- Please refer Readme for training GloRe model on Kinetics.

Final Report (code implementation, additional experiments)

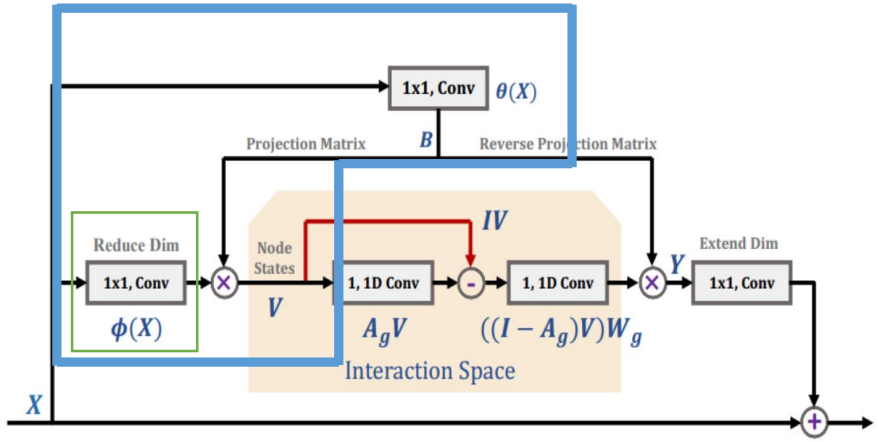
Gee Ho Kim

Seoul National University

Table of contents

- 최종 리포트를 통해 본 논문에서 제안한 GloRe Unit을 reproduce 및 코드 설명
- Fine-grained image classification 실험 과정 및 결과 (w/, w/o GloRe Unit)

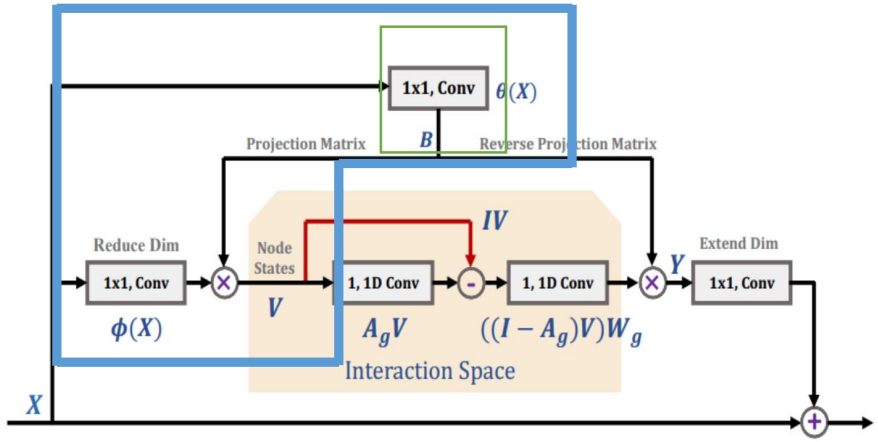
Global Reasoning (GloRe) Unit



- symbol_glore.py
- Dimension reduction by BN_AC_Conv

```python
# - - - - - - - - - - - - - - - - - - - - - - - - -
# GloRe Unit
def GloRe_Unit(data, settings, name, stride=(1,1)):
    num_in, num_mid = settings
    num_state = int(2 * num_mid)
    num_node = int(1 * num_mid)

    # reduce sampling space if is required
    if tuple(stride) == (1,1):
        x_pooled = data  # default
    else:
        x_pooled = mx.symbol.Pooling(data=data, pool_type="avg", kernel=stride, stride=stride, name=('%s_pooling' % name))

    # generate "state" for each node:
    # (N, num_in, H_sampled, W_sampled) --> (N, num_state, H_sampled, W_sampled)
    #                                   --> (N, num_state, H_sampled*W_sampled)
    x_state = BN_AC_Conv(data=x_pooled, num_filter=num_state,  kernel=(1, 1), pad=(0, 0), name=('%s_conv-state' % name))
    x_state_reshaped = mx.symbol.reshape(x_state, shape=(0, 0, -1),                       name=('%s_conv-state-reshape' % name))

    # prepare "projection" function (coordinate space -> interaction space):
    # (N, num_in, H_sampled, W_sampled) --> (N, num_node, H_sampled, W_sampled)
    #                                   --> (N, num_node, H_sampled*W_sampled)
    x_proj = BN_AC_Conv(data=x_pooled, num_filter=num_node, kernel=(1, 1), pad=(0, 0),   name=('%s_conv-proj' % name))
    x_proj_reshaped = mx.symbol.reshape(x_proj, shape=(0, 0, -1),                        name=('%s_conv-proj-reshape' % name))

    # prepare "reverse projection" function (interaction space -> coordinate space)
    # (N, num_in, H, W) --> (N, num_node, H, W)
    #                   --> (N, num_node, H*W)
    x_rproj_reshaped = x_proj_reshaped

    # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
    # Projection: coordinate space -> interaction space
    # (N, num_state, H_sampled*W_sampled) x (N, num_node, H_sampled*W_sampled)T --> (N, num_state, num_node)
    x_n_state = mx.symbol.batch_dot(lhs=x_state_reshaped, rhs=x_proj_reshaped, transpose_b=True,  name=('%s_proj' % name))
```

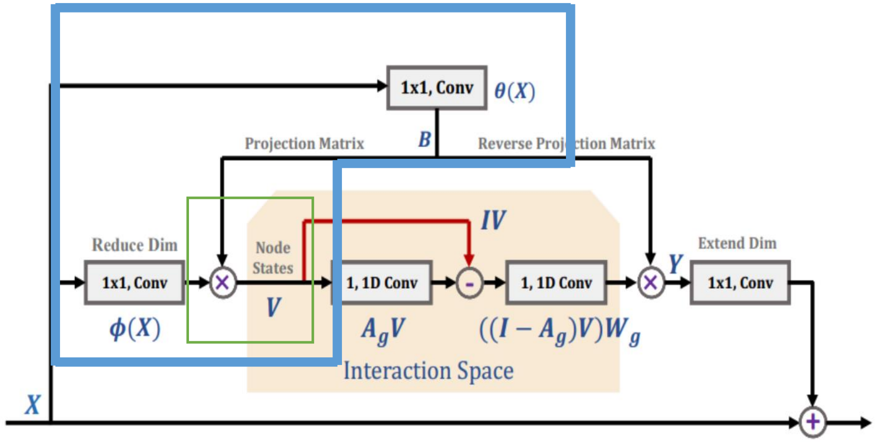
Global Reasoning (GloRe) Unit



- Generate projection matrix by BN_AC_Conv

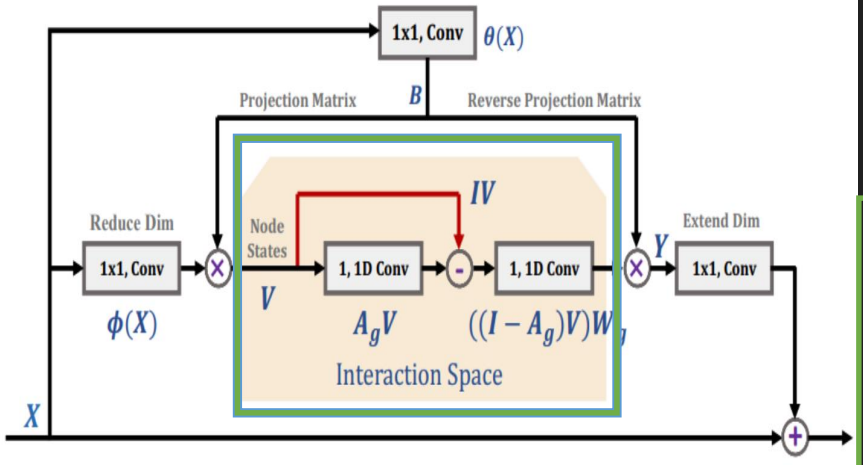
```
# - - - - - - - - - - - - - - - - - - - - - - - - -
# GloRe Unit
def GloRe_Unit(data, settings, name, stride=(1,1)):
    num_in, num_mid = settings
    num_state = int(2 * num_mid)
    num_node = int(1 * num_mid)

    # reduce sampling space if is required
    if tuple(stride) == (1,1):
        x_pooled = data  # default
    else:
        x_pooled = mx.symbol.Pooling(data=data, pool_type="avg", kernel=stride, stride=stride, name=('%s_pooling' % name))

    # generate "state" for each node:
    # (N, num_in, H_sampled, W_sampled) --> (N, num_state, H_sampled, W_sampled)
    #                                   --> (N, num_state, H_sampled*W_sampled)
    x_state = BN_AC_Conv(data=x_pooled, num_filter=num_state,  kernel=(1, 1), pad=(0, 0), name=('%s_conv-state' % name))
    x_state_reshaped = mx.symbol.reshape(x_state, shape=(0, 0, -1),                       name=('%s_conv-state-reshape' % name))

    # prepare "projection" function (coordinate space -> interaction space):
    # (N, num_in, H_sampled, W_sampled) --> (N, num_node, H_sampled, W_sampled)
    #                                   --> (N, num_node, H_sampled*W_sampled)
    x_proj = BN_AC_Conv(data=x_pooled, num_filter=num_node, kernel=(1, 1), pad=(0, 0),    name=('%s_conv-proj' % name))
    x_proj_reshaped = mx.symbol.reshape(x_proj, shape=(0, 0, -1),                         name=('%s_conv-proj-reshape' % name))

    # prepare "reverse projection" function (interaction space -> coordinate space)
    # (N, num_in, H, W) --> (N, num_node, H, W)
    #                   --> (N, num_node, H*W)
    x_rproj_reshaped = x_proj_reshaped

    # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
    # Projection: coordinate space -> interaction space
    # (N, num_state, H_sampled*W_sampled) x (N, num_node, H_sampled*W_sampled)T --> (N, num_state, num_node)
    x_n_state = mx.symbol.batch_dot(lhs=x_state_reshaped, rhs=x_proj_reshaped, transpose_b=True,  name=('%s_proj' % name))
```

Global Reasoning (GloRe) Unit




```python
# - - - - - - - - - - - - - - - - - - - - - - -
# GloRe Unit
def GloRe_Unit(data, settings, name, stride=(1,1)):
    num_in, num_mid = settings
    num_state = int(2 * num_mid)
    num_node  = int(1 * num_mid)

    # reduce sampling space if is required
    if tuple(stride) == (1,1):
        x_pooled = data  # default
    else:
        x_pooled = mx.symbol.Pooling(data=data, pool_type="avg", kernel=stride, stride=stride, name=('%s_pooling' % name))

    # generate "state" for each node:
    # (N, num_in, H_sampled, W_sampled) --> (N, num_state, H_sampled, W_sampled)
    #                                   --> (N, num_state, H_sampled*W_sampled)
    x_state = BN_AC_Conv(data=x_pooled, num_filter=num_state,  kernel=(1, 1), pad=(0, 0), name=('%s_conv-state' % name))
    x_state_reshaped = mx.symbol.reshape(x_state, shape=(0, 0, -1),                      name=('%s_conv-state-reshape' % name))

    # prepare "projection" function (coordinate space -> interaction space):
    # (N, num_in, H_sampled, W_sampled) --> (N, num_node, H_sampled, W_sampled)
    #                                   --> (N, num_node, H_sampled*W_sampled)
    x_proj = BN_AC_Conv(data=x_pooled, num_filter=num_node, kernel=(1, 1), pad=(0, 0),   name=('%s_conv-proj' % name))
    x_proj_reshaped = mx.symbol.reshape(x_proj, shape=(0, 0, -1),                        name=('%s_conv-proj-reshape' % name))

    # prepare "reverse projection" function (interaction space -> coordinate space)
    # (N, num_in, H, W) --> (N, num_node, H, W)
    #                   --> (N, num_node, H*W)
    x_rproj_reshaped = x_proj_reshaped

    # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
    # Projection: coordinate space -> interaction space
    # (N, num_state, H_sampled*W_sampled) x (N, num_node, H_sampled*W_sampled)T --> (N, num_state, num_node)
    x_n_state = mx.symbol.batch_dot(lhs=x_state_reshaped, rhs=x_proj_reshaped, transpose_b=True,  name=('%s_proj' % name))
```

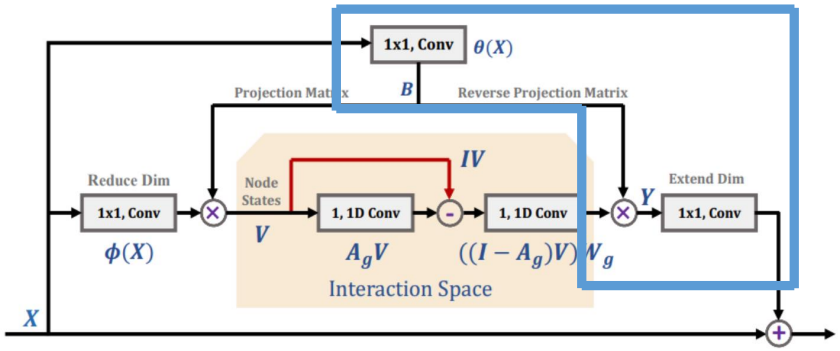
GloRe Unit




```
# -------
# Relation Reasoning
# -> propogate information: [G * X]
#     (N, num_state, num_node)
#     -permute-> (N, num_node, num_state)
#     --conv-->  (N, num_node, num_state)
#     -permute-> (N, num_state, num_node) (+ shortcut)
#     --conv-->  (N, num_state, num_node)
x_n_rel = x_n_state
x_n_rel = mx.symbol.transpose(data=x_n_rel, axes=(0,2,1),                    name=('%s_GCN-G-permute1' % name))
x_n_rel = BN_AC_Conv(data=x_n_rel, num_filter=num_node,  kernel=1, pad=0, stride=1, name=('%s_GCN-G' % name))
x_n_rel = mx.symbol.transpose(data=x_n_rel, axes=(0,2,1),                    name=('%s_GCN-G-permute2' % name))
# -> add shortcut
x_n_rel = mx.symbol.ElementWiseSum(*[x_n_state, x_n_rel],                    name=('%s_GCN-G_sum' % name))
# -> update state: [H * W]
x_n_rel = BN_AC_Conv(data=x_n_rel, num_filter=num_state, kernel=1, pad=0, stride=1, name=('%s_GCN-GHW' % name))
# -> output
x_n_state_new = x_n_rel
```

- Construction of a fully-connected graph
- Two-directional 1D convolution for the implementation of graph convolution.

GloRe Unit




```
# -------
# Reverse Projection: interaction space -> coordinate space
# (N, num_state, num_node) x (N, num_node, H*W) --> (N, num_state, H*W)
#                                                --> (N, num_state, H, W)
x_out = mx.symbol.batch_dot(lhs=x_n_state_new, rhs=x_rproj_reshaped, name=('%s_reverse-proj' % name))
x_out = mx.symbol.reshape_like(x_out, rhs=x_state,                   name=('%s_reverse-proj-reshape' % name))

# -------
# extend dimension
x_out = BN_AC_Conv(data=x_out, num_filter=num_in, kernel=(1, 1), pad=(0, 0), name=('%s_fc-2' % name))

out = mx.symbol.ElementWiseSum(*[data, x_out], name=('%s_sum' % name))
```

- Reverse Projection matrix $D = B^{\mathrm{T}}$
- BN_AC_Conv extends dimensions identical to X
- ElementWiseSum operates as residual connection.

Experiments

- Fine-grained Image classification
  - CUB200
    - Fine-grained dataset
    - 200 bird classes, 11,788 images
    - Small-scale compare to ImageNet, Kinetics
- Experimental setting (train_classifier.py)
  - Backbone
    - Resnet-18
  - Hyper-parameter
    - The number of projection matrices = 16
    - Channel reduction = 256 -> 256 (no reduction)
  - 10 epochs training
  - Training, Test batch size = 32
  - Learning rate = 0.04
  - weight decay = 5e-4, momentum=0.9 (identical to a training protocol of CUB200)

Results

- Fine-grained Image classification on CUB200
  - Top-1 accuracy after 10 epochs training
  - Res3, Res4 injection of GloRe unit after residual block 3 and 4

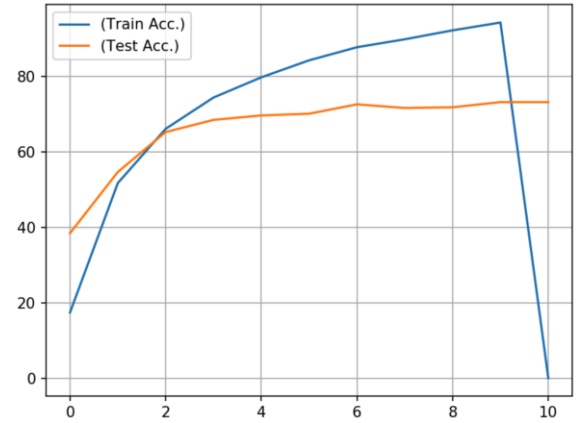
Method		Res3	Res4	#Params	Top-1
ResNet18	Baseline (w/o GloRe)			11.28M	73.23%
	GloRe	+1		11.53M	69.84%
	GloRe		+1	11.53M	Failure training

Results

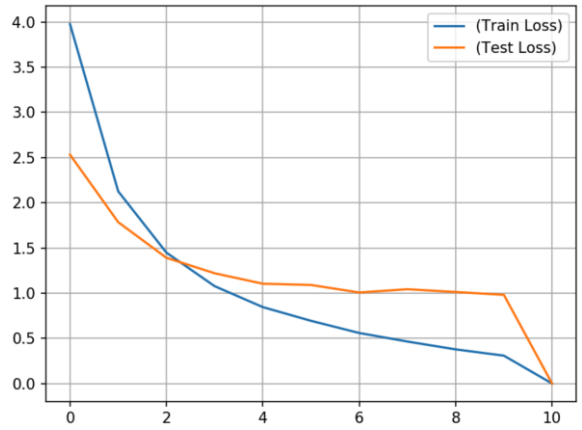
- Baseline: ResNet18 w/o GloRe unit

```
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
In[2]: runfile('/home/sonic/geeho/combinatorial_learning/semisup_comblearn/internal_noise/211_baseline_on_intranoisyset/train_classifier.py', args=['--gpu-id', '2', '--pretrained',
  '--dataset=cub200', '--epochs=10', '--momentum=0.00', '--wd=2e-4', '--lr=0.04'], wdir='/home/sonic/geeho/combinatorial_learning/semisup_comblearn/internal_noise/')
==> Preparing cub200 dataset
==> creating model resnet
    Total params: 11.28M

Epoch: [1 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.104s | Batch: 0.170s | Total: 0:00:31 | ETA: 0:00:01 | Loss: 3.9773 | top1:  17.3964 | top5:  41.2433
Processing |##############################| (182/182) Data: 0.131s | Batch: 0.158s | Total: 0:00:28 | ETA: 0:00:01 | Loss: 2.5326 | top1:  38.4363 | top5:  75.2848

Epoch: [2 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.138s | Batch: 0.199s | Total: 0:00:37 | ETA: 0:00:01 | Loss: 2.1265 | top1:  51.7547 | top5:  82.2360
Processing |##############################| (182/182) Data: 0.175s | Batch: 0.204s | Total: 0:00:37 | ETA: 0:00:01 | Loss: 1.7855 | top1:  54.6945 | top5:  84.9672

Epoch: [3 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.153s | Batch: 0.217s | Total: 0:00:40 | ETA: 0:00:01 | Loss: 1.4484 | top1:  66.1430 | top5:  90.6918
Processing |##############################| (182/182) Data: 0.166s | Batch: 0.197s | Total: 0:00:35 | ETA: 0:00:01 | Loss: 1.3912 | top1:  65.2917 | top5:  90.7491

Epoch: [4 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.160s | Batch: 0.222s | Total: 0:00:41 | ETA: 0:00:01 | Loss: 1.0772 | top1:  74.4485 | top5:  94.5354
Processing |##############################| (182/182) Data: 0.176s | Batch: 0.206s | Total: 0:00:37 | ETA: 0:00:01 | Loss: 1.2200 | top1:  68.5192 | top5:  91.4222

Epoch: [5 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.145s | Batch: 0.206s | Total: 0:00:38 | ETA: 0:00:01 | Loss: 0.8453 | top1:  79.8128 | top5:  96.8082
Processing |##############################| (182/182) Data: 0.169s | Batch: 0.197s | Total: 0:00:35 | ETA: 0:00:01 | Loss: 1.1043 | top1:  69.6928 | top5:  92.2679

Epoch: [6 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.151s | Batch: 0.212s | Total: 0:00:39 | ETA: 0:00:01 | Loss: 0.6936 | top1:  84.3249 | top5:  97.4265
Processing |##############################| (182/182) Data: 0.166s | Batch: 0.191s | Total: 0:00:34 | ETA: 0:00:01 | Loss: 1.0904 | top1:  70.1588 | top5:  92.3369

Epoch: [7 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.138s | Batch: 0.202s | Total: 0:00:37 | ETA: 0:00:01 | Loss: 0.5589 | top1:  87.8008 | top5:  98.3957
Processing |##############################| (182/182) Data: 0.166s | Batch: 0.194s | Total: 0:00:35 | ETA: 0:00:01 | Loss: 1.0070 | top1:  72.6441 | top5:  92.9410

Epoch: [8 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.148s | Batch: 0.208s | Total: 0:00:38 | ETA: 0:00:01 | Loss: 0.4644 | top1:  89.9064 | top5:  98.7801
Processing |##############################| (182/182) Data: 0.173s | Batch: 0.201s | Total: 0:00:36 | ETA: 0:00:01 | Loss: 1.0432 | top1:  71.6776 | top5:  92.4922

Epoch: [9 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.116s | Batch: 0.171s | Total: 0:00:32 | ETA: 0:00:01 | Loss: 0.3776 | top1:  92.2794 | top5:  99.4151
Processing |##############################| (182/182) Data: 0.135s | Batch: 0.160s | Total: 0:00:29 | ETA: 0:00:01 | Loss: 1.0126 | top1:  71.8674 | top5:  92.6476

Epoch: [10 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.104s | Batch: 0.163s | Total: 0:00:30 | ETA: 0:00:01 | Loss: 0.3084 | top1:  94.3516 | top5:  99.5655
Processing |##############################| (182/182) Data: 0.130s | Batch: 0.155s | Total: 0:00:28 | ETA: 0:00:01 | Loss: 0.9812 | top1:  73.2309 | top5:  93.0100
Best acc: 73.23092854677252
```

Terminal log



Top-1 accuracy



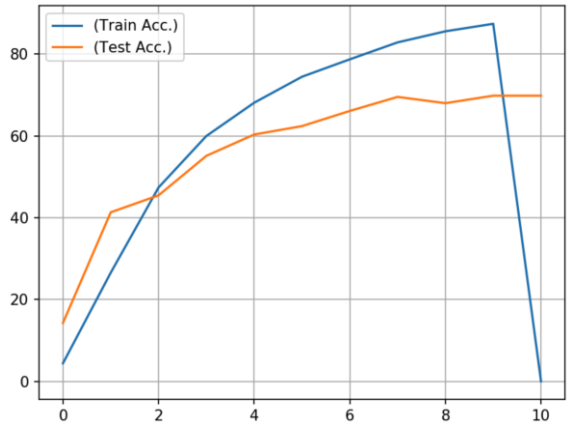
Loss

Results

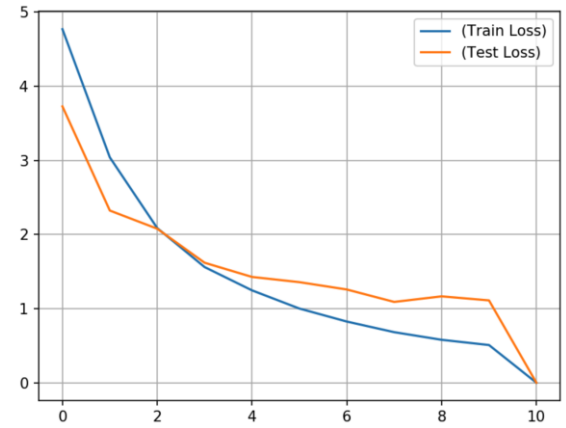
- ResNet 18 with one GloRe unit

```
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
In[2]: runfile('/home/sonic/geeho/combinatorial_learning/semisup_comblearn/internal_noise/611_glore_on_intranoisyset/train_classifier.py', args=['--dataset=cub200', '--noise-rate=0.00', '--gpu-id=0',
  '--seed=12345', '--lr=0.04', '--pretrained'], wdir='/home/sonic/geeho/combinatorial_learning/semisup_comblearn/internal_noise/')
==> Preparing cub200 dataset
==> creating model resnet
    Total params: 11.53M

Epoch: [1 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.064s | Batch: 0.124s | Total: 0:00:23 | ETA: 0:00:01 | Loss: 4.9029 | top1:  3.6096 | top5:  13.4358
Processing |##############################| (182/182) Data: 0.093s | Batch: 0.118s | Total: 0:00:21 | ETA: 0:00:01 | Loss: 3.7921 | top1:  15.4125 | top5:  41.9572

Epoch: [2 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.071s | Batch: 0.121s | Total: 0:00:22 | ETA: 0:00:01 | Loss: 3.1456 | top1:  25.1504 | top5:  57.6872
Processing |##############################| (182/182) Data: 0.093s | Batch: 0.113s | Total: 0:00:20 | ETA: 0:00:01 | Loss: 2.3761 | top1:  41.3876 | top5:  75.1122

Epoch: [3 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.069s | Batch: 0.115s | Total: 0:00:21 | ETA: 0:00:01 | Loss: 2.1314 | top1:  47.0254 | top5:  80.1303
Processing |##############################| (182/182) Data: 0.099s | Batch: 0.118s | Total: 0:00:21 | ETA: 0:00:01 | Loss: 1.9034 | top1:  51.0183 | top5:  82.6200

Epoch: [4 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.069s | Batch: 0.118s | Total: 0:00:22 | ETA: 0:00:01 | Loss: 1.5841 | top1:  59.7259 | top5:  88.4693
Processing |##############################| (182/182) Data: 0.094s | Batch: 0.114s | Total: 0:00:20 | ETA: 0:00:01 | Loss: 1.6138 | top1:  55.6783 | top5:  86.4342

Epoch: [5 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.074s | Batch: 0.124s | Total: 0:00:23 | ETA: 0:00:01 | Loss: 1.2630 | top1:  66.7279 | top5:  92.2627
Processing |##############################| (182/182) Data: 0.094s | Batch: 0.114s | Total: 0:00:20 | ETA: 0:00:01 | Loss: 1.4946 | top1:  58.8712 | top5:  87.3490

Epoch: [6 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.072s | Batch: 0.120s | Total: 0:00:22 | ETA: 0:00:01 | Loss: 1.0167 | top1:  73.2787 | top5:  94.9198
Processing |##############################| (182/182) Data: 0.093s | Batch: 0.113s | Total: 0:00:20 | ETA: 0:00:01 | Loss: 1.3451 | top1:  62.4094 | top5:  88.9713

Epoch: [7 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.071s | Batch: 0.119s | Total: 0:00:22 | ETA: 0:00:01 | Loss: 0.8314 | top1:  78.9940 | top5:  96.4071
Processing |##############################| (182/182) Data: 0.087s | Batch: 0.107s | Total: 0:00:19 | ETA: 0:00:01 | Loss: 1.3817 | top1:  62.6855 | top5:  88.7125

Epoch: [8 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.071s | Batch: 0.122s | Total: 0:00:22 | ETA: 0:00:01 | Loss: 0.7017 | top1:  82.3028 | top5:  97.2092
Processing |##############################| (182/182) Data: 0.095s | Batch: 0.116s | Total: 0:00:21 | ETA: 0:00:01 | Loss: 1.1479 | top1:  67.8288 | top5:  90.9389

Epoch: [9 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.062s | Batch: 0.113s | Total: 0:00:21 | ETA: 0:00:01 | Loss: 0.5869 | top1:  85.4947 | top5:  97.8610
Processing |##############################| (182/182) Data: 0.092s | Batch: 0.113s | Total: 0:00:20 | ETA: 0:00:01 | Loss: 1.2421 | top1:  67.1039 | top5:  89.6962

Epoch: [10 | 10] LR: 0.040000
Processing |##############################| (187/187) Data: 0.068s | Batch: 0.119s | Total: 0:00:22 | ETA: 0:00:01 | Loss: 0.5247 | top1:  87.1658 | top5:  98.6130
Processing |##############################| (182/182) Data: 0.089s | Batch: 0.110s | Total: 0:00:19 | ETA: 0:00:01 | Loss: 1.1254 | top1:  69.9862 | top5:  91.4912
Best acc: 69.98619261304798
```

Terminal log



Top-1 accuracy



Loss

Discussion

- Performance degradation about 4% on CUB200
  - Expect to capture discriminative features to classify fine-grained images
  - But, injection of the propose module results in degradation on the benchmark.
  - Addition of GloRe unit after fourth residual block of ResNet18 introduces huge loss, not trainable despite learning rate tuning, unstable.
- Future Work
  - Performance varying the number of projection matrices
    - The paper does not specify the number of projection matrices for training.
  - Injection of GloRe unit at early layers.

Conclusion

- Code Link: https://github.com/snow12345/GCN_PROJECT.git
- Environment
  - Pytorch 1.2+
- To train Resnet18 + GloRe on CUB200
  - Python train_classifier.py –gpu-id=0 –seed=0 –dataset=cub200 –lr=0.01 –pretrained –epochs=10