Iterative Visual Reasoning Beyond Convolutions

Xinlei Chen, Li-Jia Li, Li Fei-Fei and Abhinav Gupta.

CVPR 2018 (https://arxiv.org/abs/1803.11189)

Jung Hun Oh

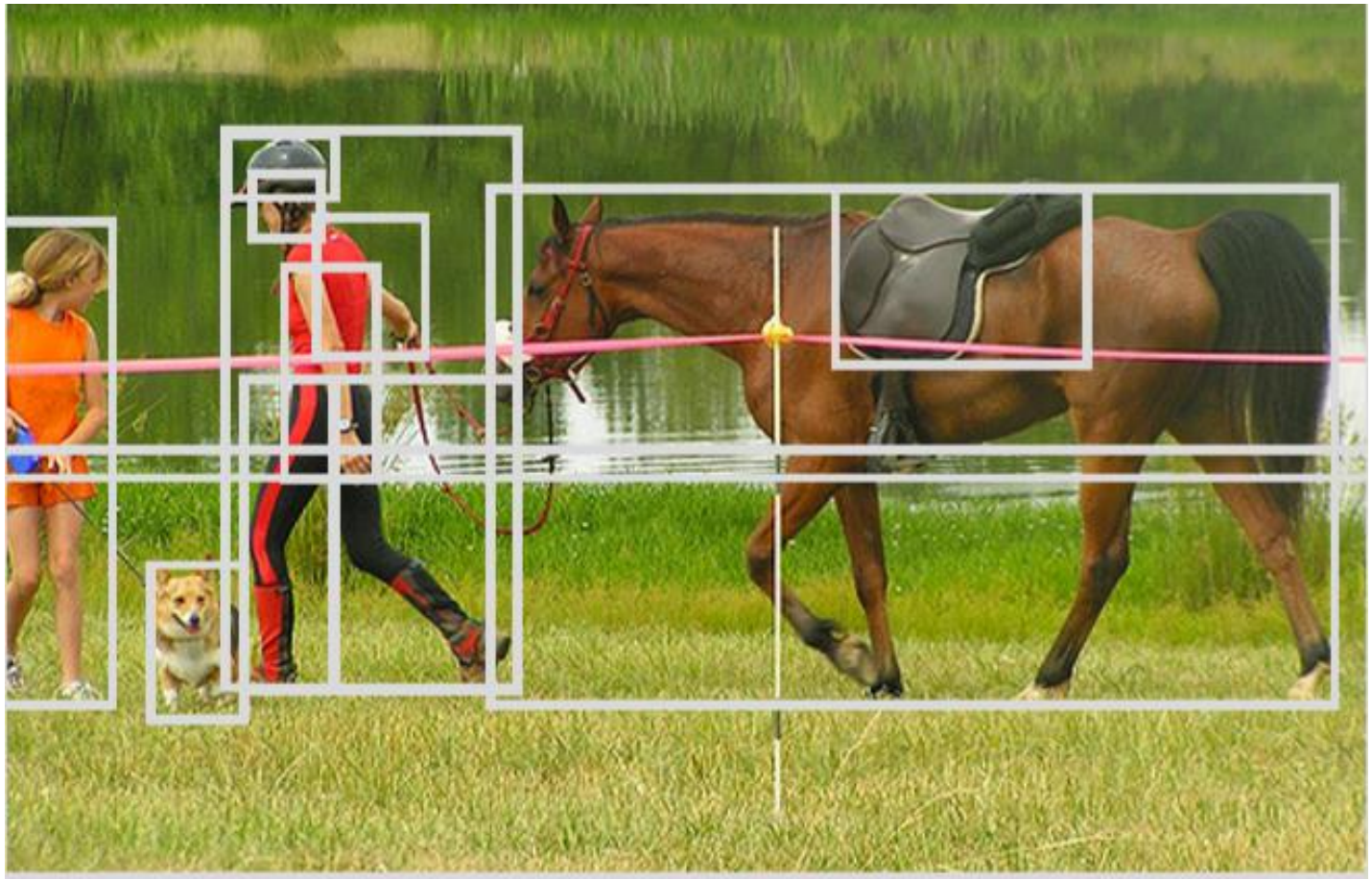
Seoul National University

Motivation

- Current visual recognition systems lack the capability to reason beyond stack of convolutional layers because of its confined receptive fields.

- By using graphs, we can use globally spatial and semantic relationships between objects.

Motivation



Spatial relationship

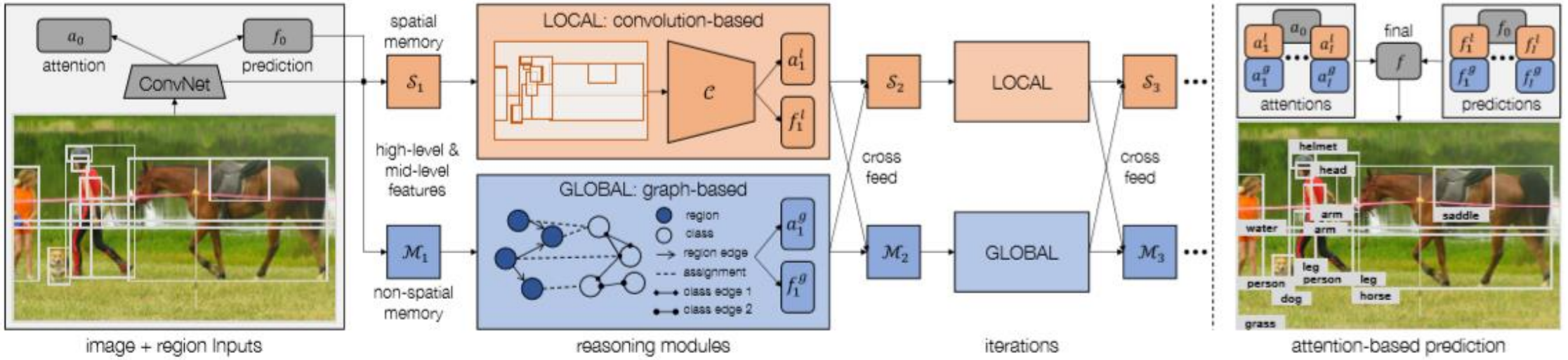
+

Semantic relationship

↓

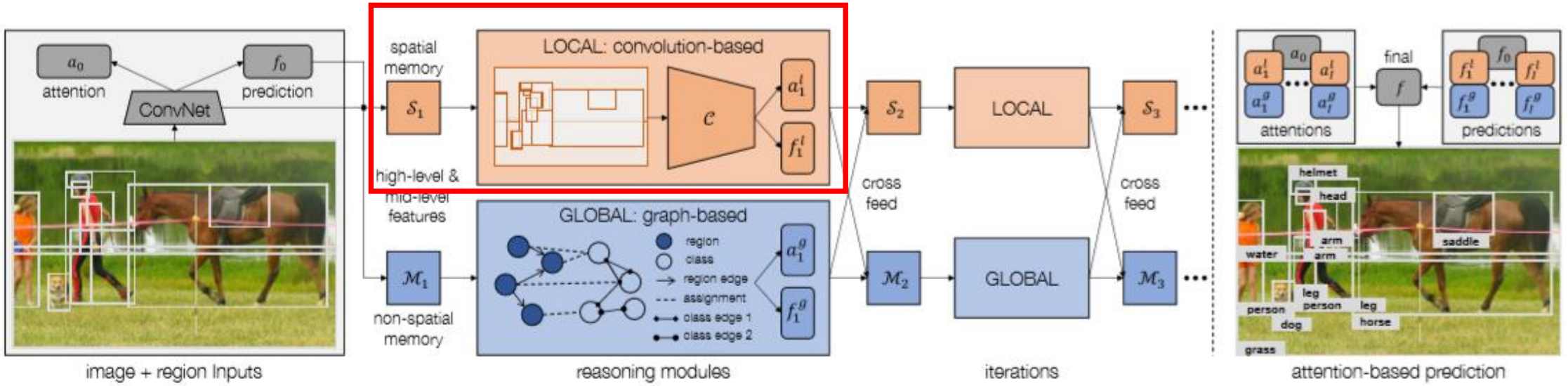
Global level reasoning

Model



- Local module: performs pixel-level reasoning using ConvNet.
- Global module: performs global-level reasoning using graphs

Local Module



- For a region of interest, crop mid-level feature and append high-level feature on it -> f_r
- Using GRU(gated recurrent unit), update spatial memory for region r

$$s'_r = u \circ s_r + (1 - u) \circ \sigma(W_f f_r + W_s(z \circ s_r) + b)$$

- u: update gate, z: reset gate

Global Module

- **Graphs in Global module**

Region to Region graphs: Spatial relationship

- Can have multiple types of edges (left/right, top/bottom …)
- Nodes: $\mathcal{N}_r$, Edges: $\mathcal{E}_{r \to r}$

$$\kappa(x) = \exp(-x/\Delta)$$

$(where\ \Delta = 50\ is\ the\ bandwidth)$

Global Module

- **Graphs in Global module**

Region to Class (class to Region): Assignment

- Assign regions to classes using soft-max score.
- Propagate beliefs from region to class or class to region.
- Edges: $e_{r \to c}$, $e_{c \to r}$

Global Module

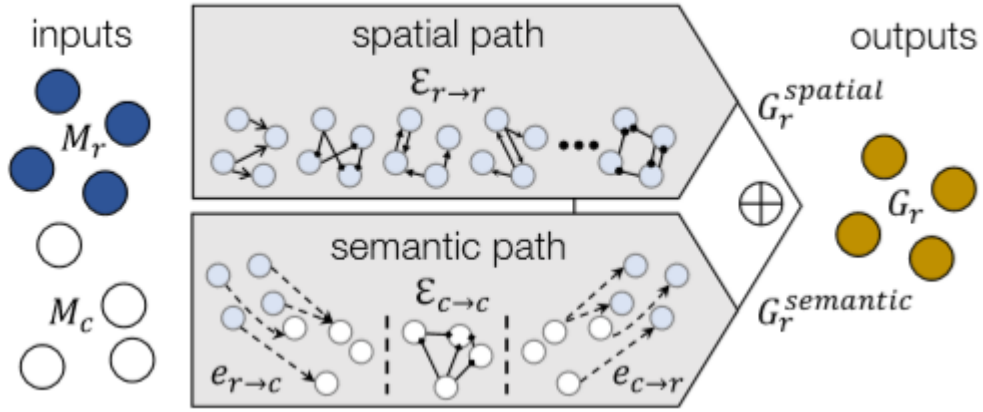
- **Graphs in Global module**

Class to Class: Semantic relationship

- Construct knowledge bases for encoding semanctic relationships between classes
- Can have multiple types of edges (is-kind-of, is-part-of …)
- Nodes: $\mathcal{N}_c$, Edges: $\mathcal{E}_{c \to c}$

Global Module

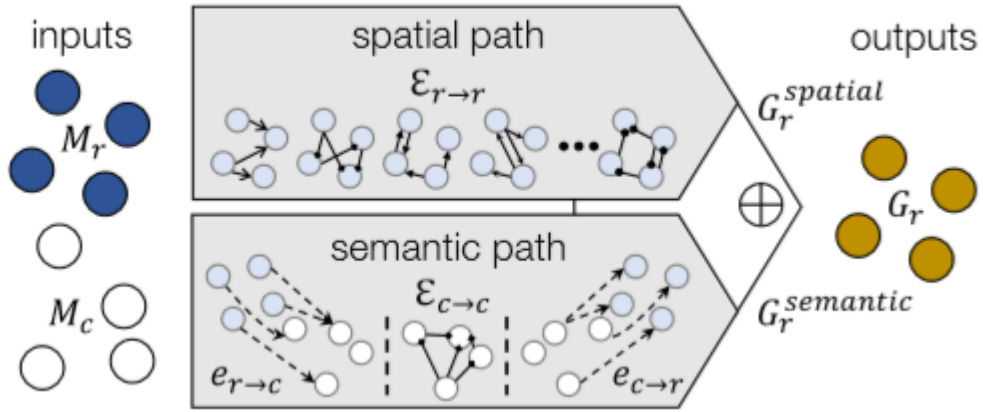
- **Graphs in Global module**



- Node features: $M_r \in \mathbb{R}^{R\times D}$, (R : # of regions, D: feature dimension)
- Class features: $M_c \in \mathbb{R}^{C\times D}$, (C: # of classes)

Global Module

- **Feature enhancement**

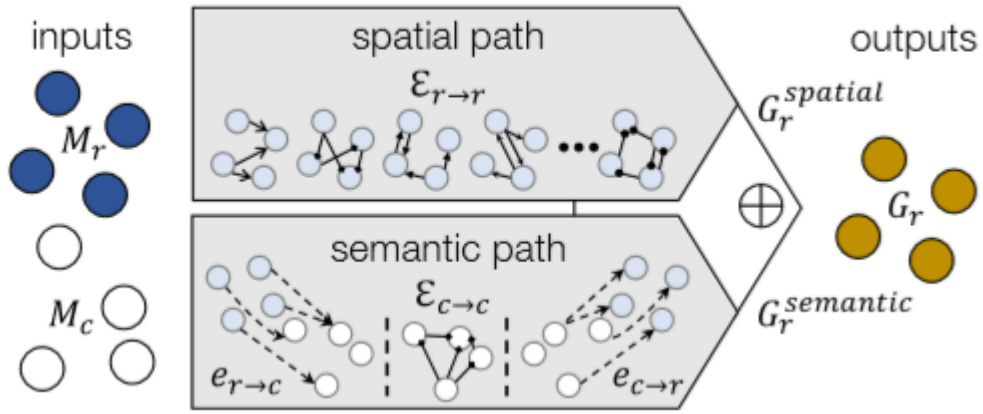


- Spatial path: enhance region features using spatial relationships

$$G_r^{spatial} = \sum_{e \in \mathcal{E}_{r\to r}} A_e M_r W_e,$$

Global Module

- **Feature enhancement**

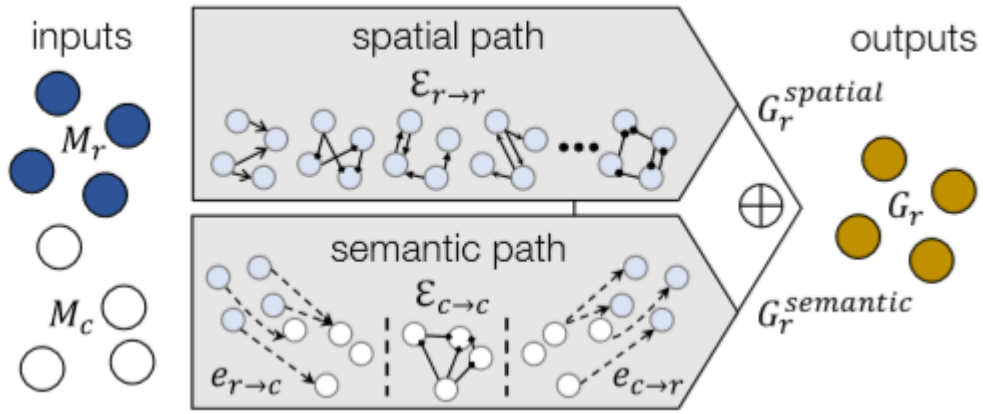


- Semantic path: enhance region and class feature using semantic relationships

$$G_c^{semantic} = \sum_{e\in\mathcal{E}_{c\to c}} A_e\sigma(A_{e_{r\to c}}M_rW_{e_{r\to c}} + M_cW_c)W_e,$$

Global Module

- **Feature enhancement**



- Final outputs

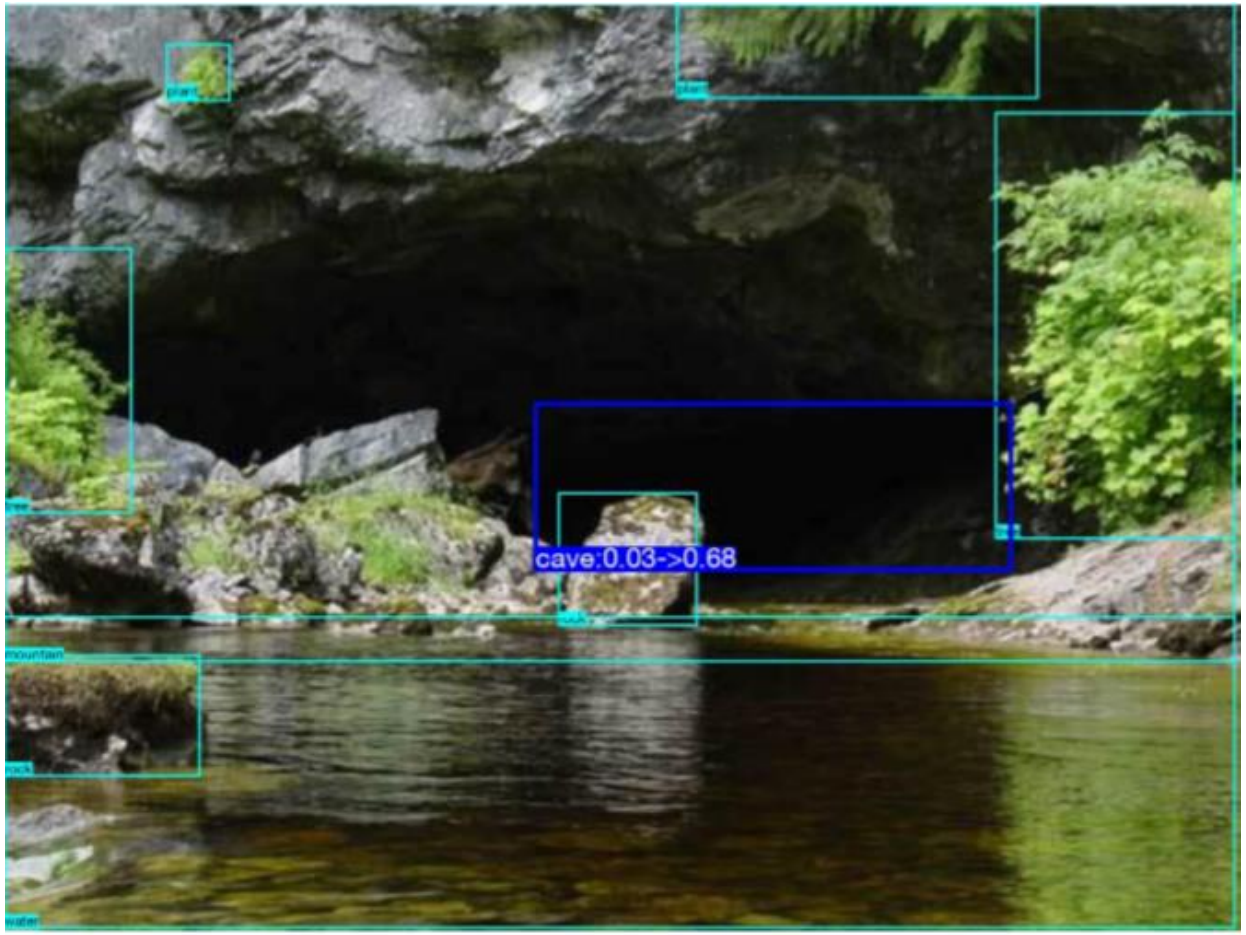
$$G_r = \sigma(G_r^{spatial} + \sigma(A_{e_{c\to r}} G_c^{semantic} W_{e_{c\to r}}))$$

Results

Table 1. Main results on ADE `test-1k` and VG `test`. AP is average precision, AC is classification accuracy. Superscripts show the improvement ∇ over the baseline.

%	Method	per-instance		per-class	
		AP^{∇}	AC^{∇}	AP^{∇}	AC^{∇}
ADE	Baseline	67.0	67.0	40.1	33.2
	w/ ResNet-101	68.2	68.3	40.8	34.4
	w/ 800-input	68.2	68.2	41.0	34.3
	Ensemble	68.7	68.8	42.9	35.3
	$Ours_{-Local}$	$71.6^{+4.6}$	$71.7^{+4.7}$	$47.9^{+7.8}$	$38.7^{+5.7}$
	$Ours_{-Global}$	$69.8^{+2.8}$	$69.8^{+2.8}$	$44.5^{+4.4}$	$36.8^{+3.6}$
	$Ours_{-Final}$	$\mathbf{72.6}^{+5.6}$	$\mathbf{72.6}^{+5.6}$	$\mathbf{48.5}^{+8.4}$	$\mathbf{39.5}^{+6.3}$
VG	Baseline	49.1	49.6	16.9	12.1
	w/ ResNet-101	50.3	50.8	18.0	**13.0**
	w/ 800-input	49.5	50.0	17.0	12.2
	w/ Ensemble	50.2	50.7	17.7	12.3
	$Ours_{-Local}$	$51.4^{+2.3}$	$51.9^{+2.3}$	$18.8^{+1.9}$	$12.8^{+0.7}$
	$Ours_{-Global}$	$50.9^{+1.8}$	$51.5^{+1.9}$	$18.3^{+1.4}$	$12.6^{+0.5}$
	$Ours_{-Final}$	$\mathbf{51.7}^{+2.6}$	$\mathbf{52.2}^{+2.6}$	$\mathbf{19.1}^{+2.2}$	$12.9^{+0.8}$

Results



conclusion

- This paper uses graph to encode spatial and semantic relationship between regions and classes and passes massage on the graph.

- Its results show that the global reasoning beyond convolutions works better.

Reproducing Report

공개한 코드가 오래된 버전 (2018년도)이어서 상당 부분을 수정하고 여러 패키지 및 cuda의 version을 맞추는 것에 많은 시간이 소요되었습니다. 또한 train에 많은 memory와 시간이 소모되어 논문에 나온 결과를 제대로 reproduce 하지 못하여 대신 code 분석과 어느정도 traing한 결과를 report 하였습니다.

Code analysis

1. In experiments/scripts/train_memory.sh run tools/trainval_memory.py

```
63  if [ ! -f ${NET_FINAL}.index ]; then
64    CUDA_VISIBLE_DEVICES=${GPU_ID} python ./tools/trainval_memory.py \
65      --weight data/imagenet_weights/${NET_BASE}.ckpt \
66      --imdb ${TRAIN_IMDB} \
67      --imdbval ${VAL_IMDB} \
68      --iters ${ITERS} \
69      --cfg experiments/cfgs/${NET}.yml \
70      --tag ${EXTRA_ARGS_SLUG} \
71      --net ${NET} \
72      --set TRAIN.STEPSIZE ${STEPSIZE} ${EXTRA_ARGS}
```

Code analysis

2. In tools/trainnal_memory.py run train_net in lib/model/train_val_memory.py

```
138    train_net(net, imdb, roidb, valroidb, output_dir, tb_dir,
139              pretrained_model=args.weight,
140              max_iters=args.max_iters)
```

Code analysis

3. In train_net run train_model in same file (line 73)

```
158 def train_net(network, imdb, roidb, valroidb, output_dir, tb_dir,
159                pretrained_model=None,
160                max_iters=40000):
161   """Train a Faster R-CNN network with memory."""
162   roidb = filter_roidb(roidb)
163   valroidb = filter_roidb(valroidb)
164 
165   tfconfig = tf.ConfigProto(allow_soft_placement=True)
166   tfconfig.gpu_options.allow_growth = True
167 
168   with tf.Session(config=tfconfig) as sess:
169     sw = MemorySolverWrapper(sess, network, imdb, roidb, valroidb,
170                              output_dir, tb_dir,
171                              pretrained_model=pretrained_model)
172     print('Solving...')
173     sw.train_model(sess, max_iters)
174     print('done solving')
```

Code analysis

4. In train_model load data and construct graph

```
73    def train_model(self, sess, max_iters):
74      # Build data layers for both training and validation set
75      self.data_layer = self.imdb.data_layer(self.roidb, self.imdb.num_classes)
76      self.data_layer_val = self.imdb.data_layer(self.valroidb, self.imdb.num_classes, random=True)
77
78      # Construct the computation graph
79      lr, train_op = self.construct_graph(sess)
80
81      # Find previous snapshots if there is any to restore from
82      lsf, nfiles, sfiles = self.find_previous()
83
84      # Initialize the variables or restore them from the last snapshot
85      if lsf == 0:
86        rate, last_snapshot_iter, stepsizes, np_paths, ss_paths = self.initialize(sess)
87      else:
88        rate, last_snapshot_iter, stepsizes, np_paths, ss_paths = self.restore(sess,
89                                                                               str(sfiles[-1]),
90                                                                               str(nfiles[-1]))
91      timer = Timer()
92      iter = last_snapshot_iter + 1
93      last_summary_iter = iter
94      last_summary_time = time.time()
95      # Make sure the lists are not empty
96      stepsizes.append(max_iters)
97      stepsizes.reverse()
98      next_stepsize = stepsizes.pop()
```

-> Load data

-> construct graph

Code analysis

5. In construct_graph in lib/model/train_val_memoey.py, construct graph for training

```
27    def construct_graph(self, sess):
28      with sess.graph.as_default():
29        # Set the random seed for tensorflow
30        tf.set_random_seed(cfg.RNG_SEED)
31        # Build the main computation graph
32        layers = self.net.create_architecture('TRAIN', self.imdb.num_classes, tag='default')
33        # Define the loss
34        loss = layers['total_loss']
35        # Set learning rate and momentum
36        lr = tf.Variable(cfg.TRAIN.RATE, trainable=False)
37        self.optimizer = tf.train.MomentumOptimizer(lr, cfg.TRAIN.MOMENTUM)
38
39        # Compute the gradients with regard to the loss
40        gvs = self.optimizer.compute_gradients(loss)
41        grad_summaries = []
42        for grad, var in gvs:
43          if 'SMN' not in var.name and 'GMN' not in var.name:
44            continue
45          grad_summaries.append(tf.summary.histogram('TRAIN/' + var.name, var))
46          if grad is not None:
47            grad_summaries.append(tf.summary.histogram('GRAD/' + var.name, grad))
48
```

-> most important part

Code analysis

6. In create_architecture in lib/net/network.py, define rois and its cls_prob (output)

```
193         rois, cls_prob = self._build_network(training)
```

Code analysis

7. In _build_network in lib/nets/network.py, compute rois and its cls_prob through several conv layers

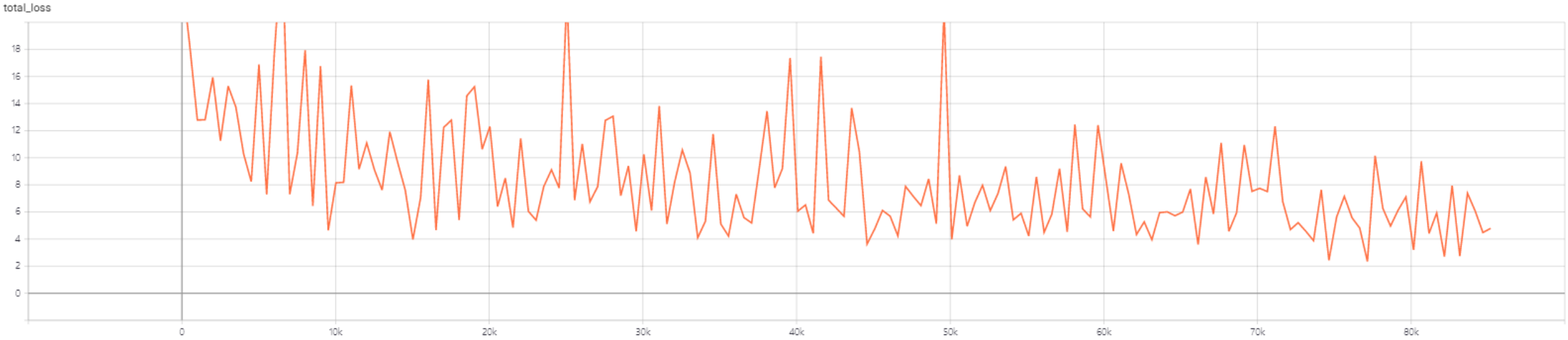
```
128   def _build_network(self, is_training=True):
129     # select initializers
130     initializer = tf.random_normal_initializer(mean=0.0, stddev=0.01)
131
132     net_conv = self._image_to_head(is_training)
133     with tf.variable_scope(self._scope, self._scope):
134       # get the region of interest
135       rois, batch_ids = self._target_layer("target")
136       # region of interest pooling
137       pool5 = self._crop_pool_layer(net_conv, rois, batch_ids, "pool5")
138
139     fc7 = self._head_to_tail(pool5, is_training)
140     with tf.variable_scope(self._scope, self._scope):
141       # region classification
142       cls_prob = self._region_classification(fc7, is_training, initializer)
143
144     self._score_summaries.update(self._predictions)
145
146     return rois, cls_prob
```

Code analysis

8. According to the graph for rois and its cls_prob, train the model iteratively in train_model function (lin 99~ 149 in train_model in lib/model/train_val_memory.py)

```
99      while iter < max_iters + 1:
100       # Learning rate
101       if iter == next_stepsize + 1:
102         # Add snapshot here before reducing the learning rate
103         self.snapshot(sess, iter)
104         rate *= cfg.TRAIN.GAMMA
105         sess.run(tf.assign(lr, rate))
106         next_stepsize = stepsizes.pop()
107
108       timer.tic()
109       # Get training data, one batch at a time
110       blobs = self.data_layer.forward()
111
112       now = time.time()
113       if iter == 1 or \
114           (now - last_summary_time > cfg.TRAIN.SUMMARY_INTERVAL and \
115           iter - last_summary_iter > cfg.TRAIN.SUMMARY_ITERS):
116         # Compute the graph with summary
117         loss_cls, total_loss, summary, gsummary = \
118           self.net.train_step_with_summary(sess, blobs, train_op, self.summary_grads)
119         self.writer.add_summary(summary, float(iter))
120         self.writer.add_summary(gsummary, float(iter+1))
121         # Also check the summary on the validation set
122         blobs_val = self.data_layer_val.forward()
123         summary_val = self.net.get_summary(sess, blobs_val)
124         self.valwriter.add_summary(summary_val, float(iter))
125         last_summary_iter = iter
126         last_summary_time = now
127       else:
128         # Compute the graph without summary
129         loss_cls, total_loss = self.net.train_step(sess, blobs, train_op)
130       timer.toc()
131
132       # Display training information
133       if iter % (cfg.TRAIN.DISPLAY) == 0:
134         print('iter: %d / %d, total loss: %.6f\n >>> loss_cls: %.6f\n >>> lr: %f' % \
135               (iter, max_iters, total_loss, loss_cls, lr.eval()))
136         print('speed: {:.3f}s / iter'.format(timer.average_time))
137
138       # Snapshotting
139       if iter % cfg.TRAIN.SNAPSHOT_ITERS == 0:
140         last_snapshot_iter = iter
141         ss_path, np_path = self.snapshot(sess, iter)
142         np_paths.append(np_path)
143         ss_paths.append(ss_path)
144
145         # Remove the old snapshots if there are too many
146         if len(np_paths) > cfg.TRAIN.SNAPSHOT_KEPT:
147           self.remove_snapshot(np_paths, ss_paths)
148
149       iter += 1
```

Reproduced Results



Total validation loss

(Test 결과는 또 다른 오류가 발생하여 얻지 못했습니다.)

Reproduced Results

(training 과정)

Reproduced Results

Git link: https://github.com/JungHunOh/gcn_project.git

Requirements: tensorflow 1.15, cuda 10.0 etc.