Factorizable Net: An Efficient Subgraph-based Framework for Scene Graph Generation

Yikang Li, Wanli Ouyang, Bolei Zhou, Jianping Shi, Chao Zhang, Xiaogang Wang

ECCV 2018

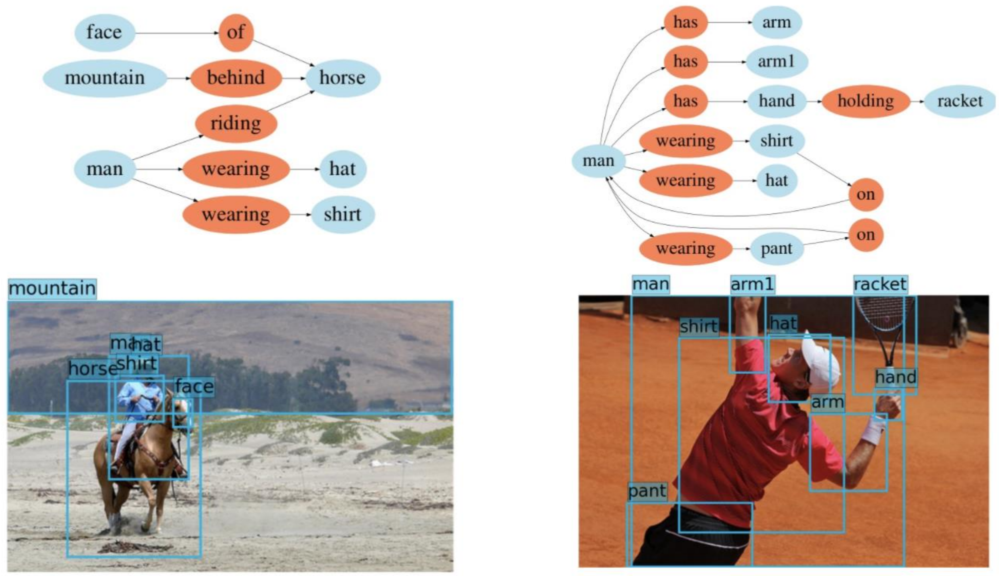
Sungeun Kim

2019-35278

Seoul National University

Scene Graph

- A set of objects node and relationships node(or edge)
  - Object node : representation of object proposal
  - Relationship node : representation of the union of boxes of the two objects
- *< subject– predicate – object >*



[Xu et al , CVPR 2017]

The problem

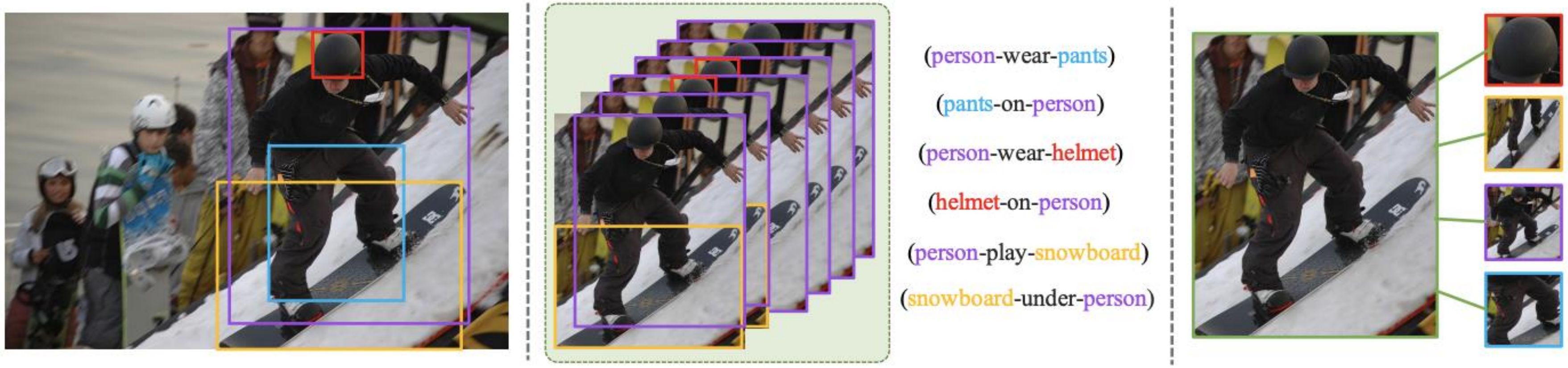
- For N object proposals, there are totally $N(N-1)$ potential relations.
- More object proposals will bring higher recall
- Number of objects increase → relationships scale quadratically → quickly becoming impractical

Motivation

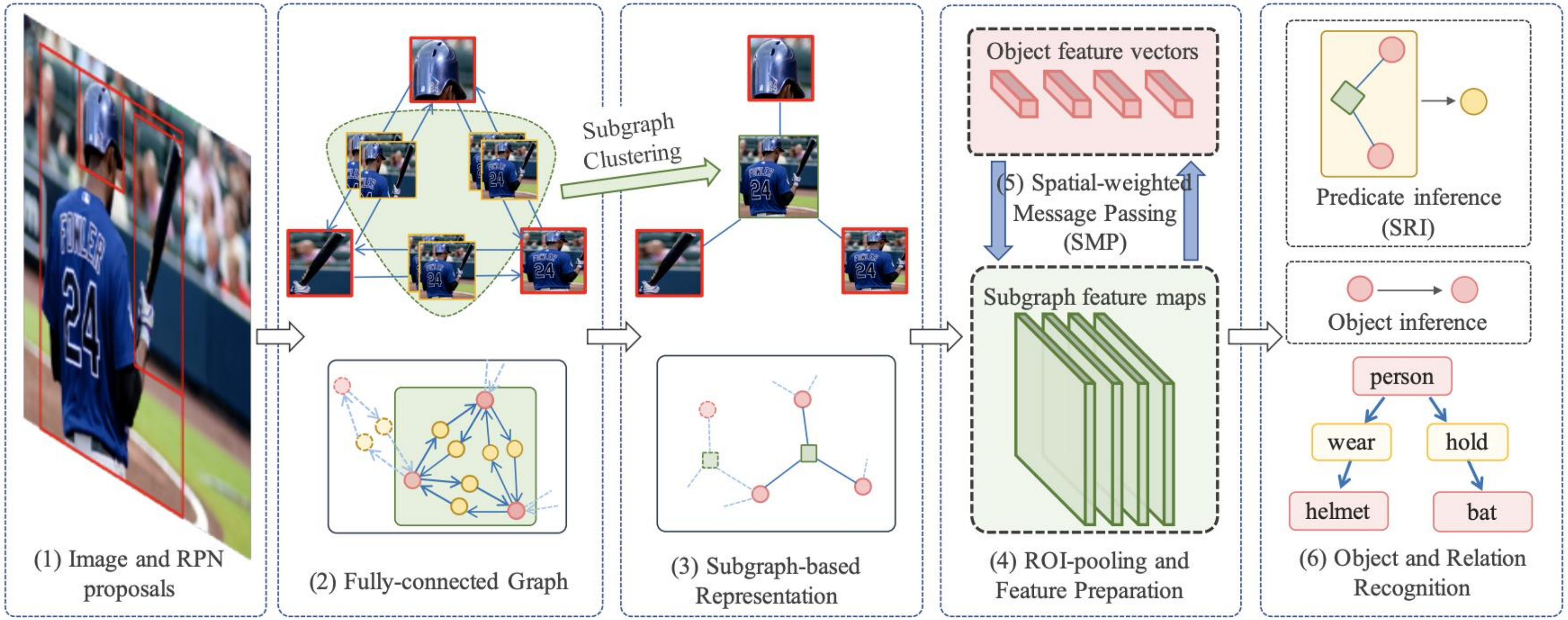
- Multiple relationship features can refer to some highly-overlapped regions
- A subgraph-based scene graph generation approach

→ The object pairs referring to the similar interacting regions are clustered into a subgraph and share the phrase representation : subgraph feature

→ speed up the model , allow us to more object proposals



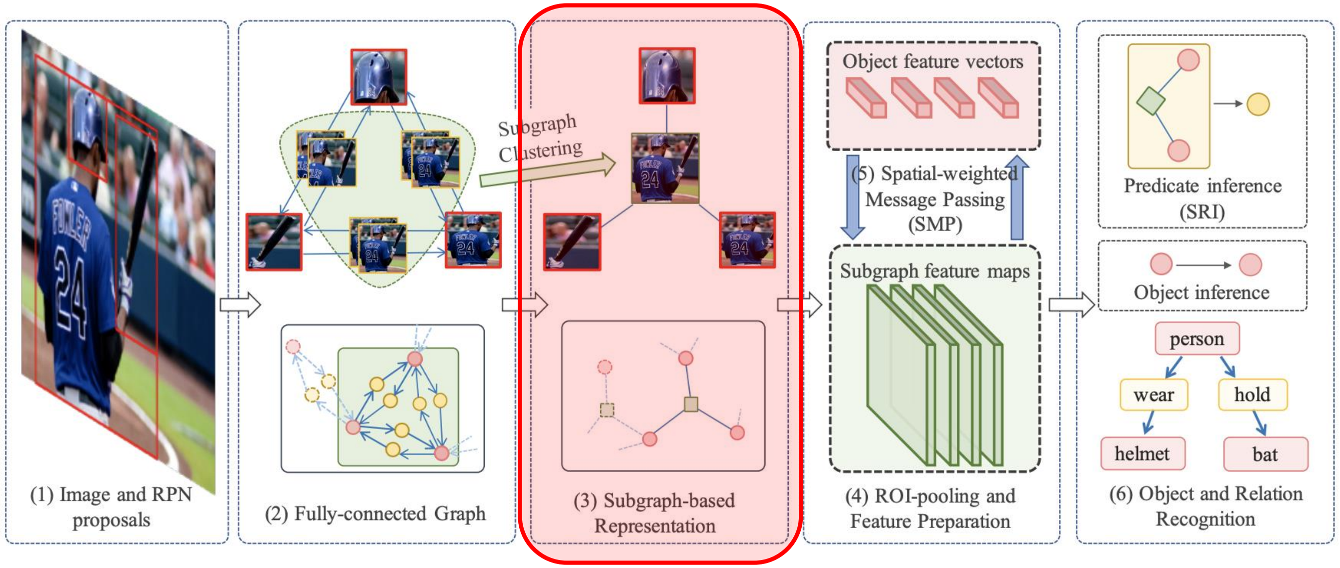
Overview of F-Net



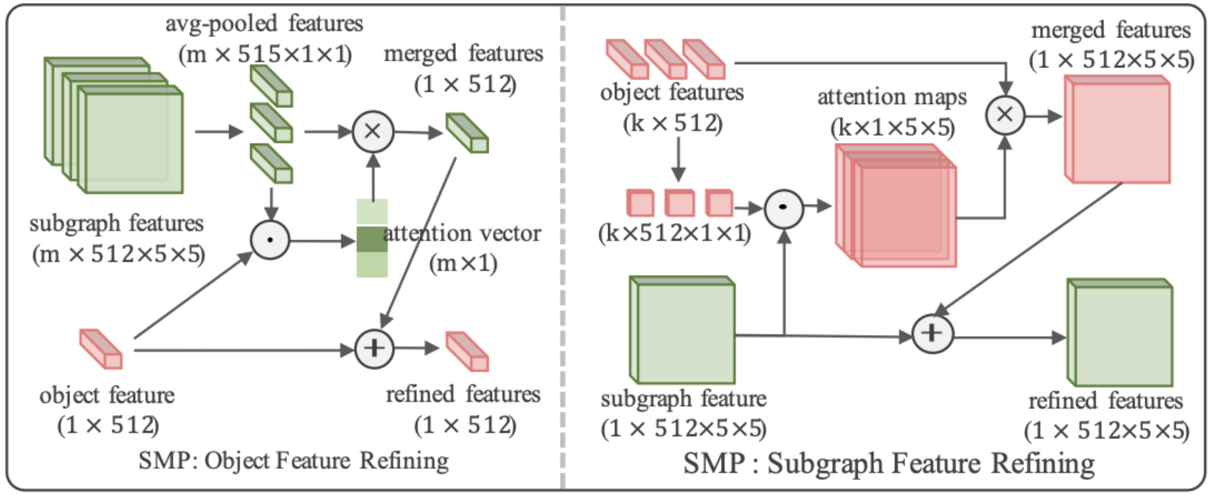
(1) Generate object region proposals with RPN
(2) Establish the fully-connected graph
(3) Cluster the fully-connected graph into several subgraphs : factorized connection graph
(4) ROI pools the objects and subgraph features and transform them into feature vectors and 2-D feature maps
(5) Jointly refine the object and subgraph features by passing message
(6) Recognize the object and relations label

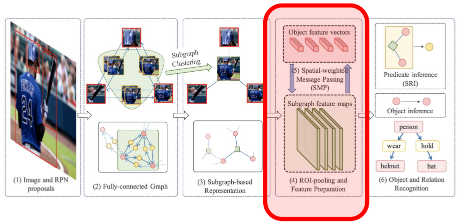
Factorized Connection Graph Generation

- Relation Confidence score = product of scores of the two object proposals
- With confidence score and bounding box location, non-maximum-suppression (NMS) can be applied.
- It does not prune the edges, but represent relationship candidates in a different form.



Spatial-weighted Message Passing (SMP)





- Pass message from subgraphs to objects

$$\tilde{\mathbf{s}}_i = \sum_{\mathbf{S_k} \in \mathbb{S}_i} p_i(\mathbf{S}_k) \cdot \mathbf{s}_k \longleftarrow \text{Subgraph feature map}$$

$$p_i(\mathbf{S}_k) = \frac{\exp\left(\mathbf{o}_i \cdot \mathrm{FC}^{(att\_s)}\left(\mathrm{ReLU}\left(\mathbf{s}_k\right)\right)\right)}{\sum_{\mathbf{S_k} \in \mathbb{C}_i} \exp\left(\mathbf{o}_i \cdot \mathrm{FC}^{(att\_s)}\left(\mathrm{ReLU}\left(\mathbf{s}_k\right)\right)\right)}$$

Attention vector

$$\hat{\mathbf{o}}_i = \mathbf{o}_i + \mathrm{FC}^{(s \to o)}\left(\mathrm{ReLU}\left(\tilde{\mathbf{s}}_i\right)\right)$$

- Pass message from objects to subgraphs

object feature vector

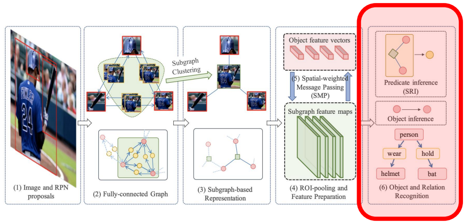
$$\tilde{\mathbf{O}}_k(x, y) = \sum_{\mathbf{o_i} \in \mathbb{O}_k} \mathbf{P}_k(\mathbf{o}_i)(x, y) \cdot \mathbf{o}_i$$

$$\mathbf{P}_k(\mathbf{o}_i)(x, y) = \frac{\exp\left(\mathrm{FC}^{(att\_o)}\left(\mathrm{ReLU}\left(\mathbf{o}_i\right)\right) \cdot \mathbf{S}_k(x, y)\right)}{\sum_{\mathbf{S_k} \in \mathbb{C}_i} \exp\left(\mathrm{FC}^{(att\_o)}\left(\mathrm{ReLU}\left(\mathbf{o}_i\right)\right) \cdot \mathbf{S}_k(x, y)\right)}$$

Attention map

$$\hat{\mathbf{S}}_k = \mathbf{S}_k + \mathrm{Conv}^{(o \to s)}\left(\mathrm{ReLU}\left(\tilde{\mathbf{O}}_k\right)\right)$$

Spatial-sensitive Relation Inference (SRI)



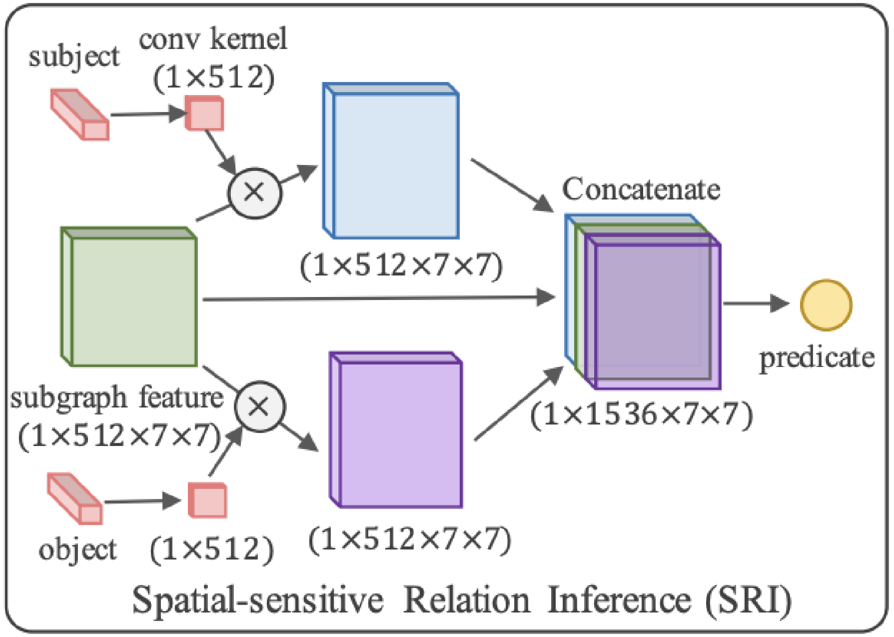
- the object features → object categories
- The subject and object features+ subgraph feature

  → relationship categories

$$\mathbf{p}^{\langle i,k,j \rangle} = \mathbf{f}\left(\mathbf{o}_i, \mathbf{S}_k, \mathbf{o}_j\right)$$

$$\mathbf{S}_k^{(i)} = \mathrm{FC}\left(\mathrm{ReLU}\left(\mathbf{o}_i\right)\right) \otimes \mathrm{ReLU}\left(\mathbf{S}_k\right)$$

$$\mathbf{p}^{\langle i,k,j \rangle} = \mathrm{FC}^{(p)}\left(\mathrm{ReLU}\left(\left[\mathbf{S}_k^{(i)}; \mathbf{S}_k; \mathbf{S}_k^{(j)}\right]\right)\right)$$



Quantitative comparison

Dataset	Model	PhrDet		SGGen		Speed
		Rec@50	Rec@100	Rec@50	Rec@100	
VRD [37]	LP [37]	16.17	17.03	13.86	14.70	1.18*
	ViP-CNN [34]	22.78	27.91	17.32	20.01	0.78
	DR-Net [6]	19.93	23.45	17.73	20.88	2.83
	ILC [42]	16.89	20.70	15.08	18.37	2.70**
	Ours Full:1-SMP	25.90	30.52	18.16	21.04	**0.45**
	Ours Full:2-SMP	**26.03**	**30.77**	**18.32**	**21.20**	0.55
VG-MSDN [28, 35]	ISGG [58]	15.87	19.45	8.23	10.88	1.64
	MSDN [35]	19.95	24.93	10.72	14.22	3.56
	Ours-Full: 2-SMP	**22.84**	**28.57**	**13.06**	**16.47**	**0.55**
VG-DR-Net [6, 28]	DR-Net [6]	23.95	27.57	**20.79**	23.76	2.83
	Ours-Full: 2-SMP	**26.91**	**32.63**	19.88	**23.95**	**0.55**

* Only consider the post-processing time given the CNN features and object detection results. ** As reported in [42], it takes about 45 minutes to test 1000 images on single K80 GPU.

ID	SubGraph	#SMP	2-D	SRI	#Boxes	PhrDet		SGGen		Speed
						R@50	R@100	R@50	R@100	Test time(s/img)
0	-	0	-	-	64	16.92	21.04	8.52	10.81	0.65
1	✓	0	-	-	64	16.50	20.79	8.49	10.33	0.18
2	✓	0	-	-	200	18.71	22.77	9.73	12.02	0.20
3	✓	0	✓	-	200	19.09	22.88	9.90	12.08	0.32
4	✓	1	✓	-	200	20.48	25.69	11.62	14.55	0.42
5	✓	1	✓	✓	200	22.54	28.31	12.83	16.12	0.44
6	✓	2	✓	✓	200	22.84	28.57	13.06	16.47	0.55

PhrDet : the performance for recognizing two objects, relation given image, object proposals
SGGen : recall of relationship triplets given image

Code Analysis

- Github path : https://github.com/sekimsekim/GCN_practice. (refer to author's code)
- Environment : Python 2.7 , cuda 9.0
  - conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
  - Requirement packages
  - Set CUDA_PATH , export CUDA_HOME=/usr/local/cuda-9.0
  - cd lib ; make all (NMS, RPN module compile)
  - export PYTHONPATH=/home2/sungeun.kim/SceneGraph/FNet_sekim
  - Pre_train model
    https://drive.google.com/uc?id=11zKRr2OF5oclFL47kjFYBOxScotQzArX&export=download
  - Download VG dataset
    Edit options/data.yaml : modify dataset dir path
    lib/datasets/visual_genome_loader.py : change dataset dir path

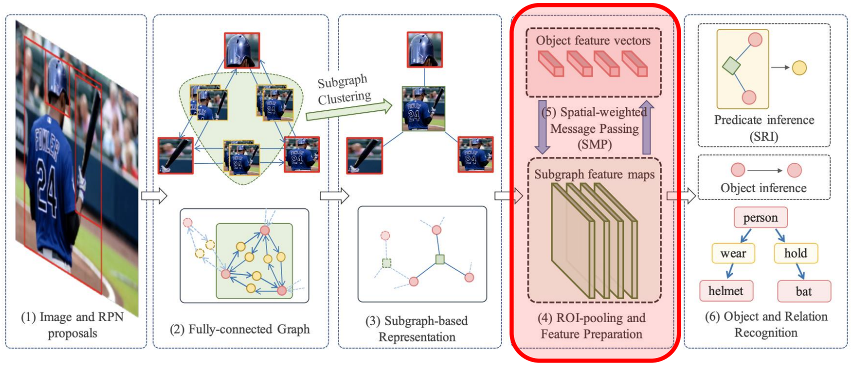
```
certifi==2019.11.28
cffi==1.14.0
click==7.1.2
cycler==0.10.0
Cython==0.29.20
easydict==1.9
future==0.18.2
h5py==2.10.0
kiwisolver==1.1.0
matplotlib==2.2.5
mkl-fft==1.0.15
mkl-random==1.1.0
mkl-service==2.3.0
numpy==1.16.6
olefile==0.46
opencv-python==4.2.0.32
Pillow==6.2.2
pycparser==2.20
pyparsing==2.4.7
python-dateutil==2.8.1
pytz==2020.1
PyYAML==5.3.1
scipy==1.2.3
six==1.15.0
subprocess32==3.5.4
torch==1.1.0
torchvision==0.2.2
tqdm==4.19.9
```

Minor fixes

- models/RPN/utils.py def build_loss(rpn_cls_score_reshape, rpn_bbox_pred, rpn_data):
  - Set same dimension because torch size of rpn_bbox_pred is different to the input size of rpn_bbox_targets.
    Ex: torch.Size([100, 37, 49])) , (torch.Size([1, 100, 37, 49])).
  - Change Loss function option

```python
    # box loss
    rpn_bbox_targets, rpn_bbox_inside_weights, rpn_bbox_outside_weights = rpn_data[1:]
    rpn_bbox_targets = torch.mul(rpn_bbox_targets, rpn_bbox_inside_weights)
    rpn_bbox_pred = torch.mul(rpn_bbox_pred, rpn_bbox_inside_weights)
    #sungeun
    rpn_bbox_pred = torch.squeeze(rpn_bbox_pred)
    # rpn_loss_box = F.smooth_l1_loss(rpn_bbox_pred, rpn_bbox_targets, size_average=False)/(fg_cnt + 1e-4)
    # print(rpn_bbox_pred.shape)
    # print(rpn_bbox_targets.shape)
    rpn_loss_box = F.smooth_l1_loss(rpn_bbox_pred, rpn_bbox_targets,  reduction='sum') / (fg_cnt + 1e-4)
    # print rpn_loss_box
    # print(type(rpn_loss_box))
```

Spatial-weighted Message Passing (SMP) code



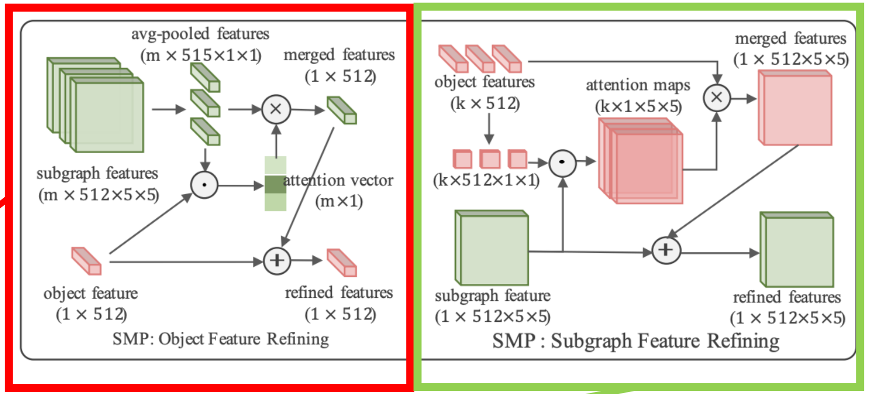
models/modules/factor_updating_structure_v3.py

class factor_updating_structure(nn.Module):

```python
def forward(self, feature_obj, feature_region, mat_object, mat_region):

    self.timer_r2o.tic()
    feature_region2object = self.region_to_object(feature_obj, feature_region, mat_object)
    # Transform the features
    out_feature_object = feature_obj + self.transform_region2object(feature_region2object)
    self.timer_r2o.toc()


    self.timer_o2r.tic()
    # gather the attentioned features
    feature_object2region = self.object_to_region(feature_region, feature_obj, mat_region)
    # Transform the features
    out_feature_region = feature_region + self.transform_object2region(feature_object2region)
    self.timer_o2r.toc()

    if TIME_IT:
        print('[MPS Timing:]')
        print('\t[R2O]: {0:.3f} s'.format(self.timer_r2o.average_time))
        print('\t[O2R]: {0:.3f} s'.format(self.timer_o2r.average_time))

    return out_feature_object, out_feature_region
```

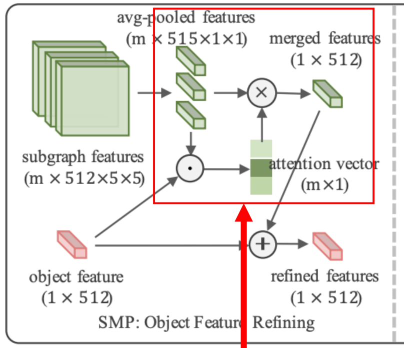
$$\hat{\mathbf{o}}_i = \mathbf{o}_i + \mathrm{FC}^{(s\to o)}\left(\mathrm{ReLU}\left(\tilde{\mathbf{s}}_i\right)\right)$$

$$\hat{\mathbf{S}}_k = \mathbf{S}_k + \mathrm{Conv}^{(o\to s)}\left(\mathrm{ReLU}\left(\tilde{\mathbf{O}}_k\right)\right)$$



① Pass message from subgraphs to object

```python
def region_to_object(self, feat_obj, feat_region, select_mat):
    feat_obj_att = self.att_region2object_obj(feat_obj)
    feat_reg_att = self.att_region2object_reg(feat_region).transpose(1, 3) # transpose the [channel] to the last
    feat_region_transposed = feat_region.transpose(1, 3)
    C_att = feat_reg_att.size(3)
    C_reg = feat_region_transposed.size(3)

    feature_data = []
    transfer_list = np.where(select_mat > 0)
    for f_id in range(feat_obj.size(0)):
        assert len(np.where(select_mat[f_id, :] > 0)[0]) > 0, "Something must be wrong. Please check the code."
        source_indices = transfer_list[1][transfer_list[0] == f_id]
        source_indices = Variable(torch.from_numpy(source_indices).type(torch.cuda.LongTensor), requires_grad=False)
        feat_region_source = torch.index_select(feat_region_transposed, 0, source_indices)
        feature_data.append(self._attention_merge(feat_obj_att[f_id],
                            torch.index_select(feat_reg_att, 0, source_indices).view(-1, C_att),
                            feat_region_source.view(-1, C_reg),))
    return torch.stack(feature_data, 0)
```



$$p_i(\mathbf{S}_k) = \frac{\exp\left(\mathbf{o}_i \cdot \mathrm{FC}^{(att\_s)}\left(\mathrm{ReLU}\left(\mathbf{s}_k\right)\right)\right)}{\sum_{\mathbf{S}_k \in \mathbf{C}_i} \exp\left(\mathbf{o}_i \cdot \mathrm{FC}^{(att\_s)}\left(\mathrm{ReLU}\left(\mathbf{s}_k\right)\right)\right)}$$

$$\tilde{\mathbf{s}}_i = \sum_{\mathbf{S}_k \in \mathbb{S}_i} p_i(\mathbf{S}_k) \cdot \mathbf{s}_k$$

```python
def _attention_merge(reference, query, features):$
    '''
    input:
        reference: vector [C] | [C x H x W]
        query: batched vectors [B x C] | [B x C x 1 x 1]
    output:
        merged message vector: [C] or [C x H x W]
    '''
    C = query.size(1)
    assert query.size(1) == reference.size(0)
    similarity = torch.sum(query * reference.unsqueeze(0), dim=1, keepdim=True) / np.sqrt(C + 1e-10)
    prob = F.softmax(similarity, dim=0)
    weighted_feature = torch.sum(features * prob, dim=0, keepdim=False)
    return weighted_feature
```

② Pass message from objects to subgraph

```
def region_to_object(self, feat_obj, feat_region, select_mat):
    feat_obj_att = self.att_region2object_obj(feat_obj)
    feat_reg_att = self.att_region2object_reg(feat_region).transpose(1, 3) # transpose the [channel] to the last
    feat_region_transposed = feat_region.transpose(1, 3)
    C_att = feat_reg_att.size(3)
    C_reg = feat_region_transposed.size(3)

    feature_data = []
    transfer_list = np.where(select_mat > 0)
    for f_id in range(feat_obj.size(0)):
        assert len(np.where(select_mat[f_id, :] > 0)[0]) > 0, "Something must be wrong. Please check the code."
        source_indices = transfer_list[1][transfer_list[0] == f_id]
        source_indices = Variable(torch.from_numpy(source_indices).type(torch.cuda.LongTensor), requires_grad=False)
        feat_region_source = torch.index_select(feat_region_transposed, 0, source_indices)
        feature_data.append(self._attention_merge(feat_obj_att[f_id],
                            torch.index_select(feat_reg_att, 0, source_indices).view(-1, C_att),
                            feat_region_source.view(-1, C_reg),))
    return torch.stack(feature_data, 0)
```

$$\mathbf{P}_k(\mathbf{o}_i)(x,y) = \frac{\exp\left(\mathrm{FC}^{(att\_o)}\left(\mathrm{ReLU}\left(\mathbf{o}_i\right)\right) \cdot \mathbf{S}_k(x,y)\right)}{\sum_{\mathbf{S_k} \in \mathbb{C}_i} \exp\left(\mathrm{FC}^{(att\_o)}\left(\mathrm{ReLU}\left(\mathbf{o}_i\right)\right) \cdot \mathbf{S}_k(x,y)\right)}$$

$$\tilde{\mathbf{O}}_k(x,y) = \sum_{\mathbf{o_i} \in \mathbb{O}_k} \mathbf{P}_k(\mathbf{o}_i)(x,y) \cdot \mathbf{o}_i$$

Training flow

models/HDN_v2/factorizable_network_v4.py

class Factorizable_network(nn.Module):

```python
def forward(self, im_data, im_info, gt_objects=None, gt_relationships=None, rpn_anchor_targets_obj=None):

    assert im_data.size(0) == 1, "Only support Batch Size equals 1"
    base_timer = Timer()
    mps_timer = Timer()
    infer_timer = Timer()
    base_timer.tic()
    # Currently, RPN support batch but not for MSDN
    features, object_rois, rpn_losses = self.rpn(im_data, im_info, rpn_data=rpn_anchor_targets_obj)
    if self.training:
        roi_data_object, roi_data_predicate, roi_data_region, mat_object, mat_phrase, mat_region = \
            self.proposal_target_layer(object_rois, gt_objects[0], gt_relationships[0], self.n_classes_obj)
        object_rois = roi_data_object[1]
        region_rois = roi_data_region[1]
    else:
        object_rois, region_rois, mat_object, mat_phrase, mat_region = self.graph_construction(object_rois,)
    # roi pool
    pooled_object_features = self.roi_pool_object(features, object_rois).view(len(object_rois), -1)
    pooled_object_features = self.fc_obj(pooled_object_features)
    # print 'fc7_object.std', pooled_object_features.data.std()

    pooled_region_features = self.roi_pool_region(features, region_rois)
    pooled_region_features = self.fc_region(pooled_region_features)

    bbox_object = self.bbox_obj(F.relu(pooled_object_features))
    base_timer.toc()

    mps_timer.tic()
    for i, mps in enumerate(self.mps_list):
        pooled_object_features, pooled_region_features = \
            mps(pooled_object_features, pooled_region_features, mat_object, mat_region)
    mps_timer.toc()

    infer_timer.tic()
    pooled_phrase_features = self.phrase_inference(pooled_object_features, pooled_region_features, mat_phrase)
    infer_timer.toc()

    cls_score_object = self.score_obj(F.relu(pooled_object_features))
    cls_prob_object = F.softmax(cls_score_object, dim=1)
    cls_score_predicate = self.score_pred(F.relu(pooled_phrase_features))
    cls_prob_predicate = F.softmax(cls_score_predicate, dim=1)
```

Extract object features by RPN

Build graph

Spatial-weighted message passing

Predict relationship

Predict object class

Train result

```
❯❯❯ cat ./scripts/train_vg_msdn.sh
#!/usr/bin/env bash

python train_FN.py --dataset_option=normal \
--path_opt options/models/VG-MSDN.yaml --rpn output/RPN.h5
```

```
sungeun.kim@duchamp /home2/sungeun.kim/SceneGraph/FactorizableNet :master conda:FactorizableNet  36s
❯❯❯ srun1 ./scripts/train_vg_msdn.sh
train_FN.py:133: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default L
afe. Please read https://msg.pyyaml.org/load for full details.
  options_yaml = yaml.load(handle)
train_FN.py:136: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default L
afe. Please read https://msg.pyyaml.org/load for full details.
  data_opts = yaml.load(f)
## args
{'MPS_iter': None,
 'clip_gradient': True,
 'dataset_option': 'normal',
 'dir_logs': None,
 'dropout': None,
 'epochs': None,
 'eval_epochs': 1,
 'evaluate': False,
 'evaluate_object': False,
 'infinite': False,
 'iter_size': 1,
 'learning_rate': None,
 'loss_weight': True,
 'model_name': None,
 'nms': -1.0,
 'optimize_MPS': False,
 'optimizer': None,
 'path_opt': 'options/models/VG-MSDN.yaml',
 'pretrained_model': None,
```

```
Epoch: [0][37000/46164] Batch_Time:  0.590      FRCNN Loss:  6.5436     RPN Loss:  1.1580
         [object] loss_cls_obj: 1.5893 loss_reg_obj: 2.0827 loss_cls_rel: 1.7186
Epoch: [0][38000/46164] Batch_Time:  0.590      FRCNN Loss:  6.5214     RPN Loss:  1.1619
         [object] loss_cls_obj: 1.5832 loss_reg_obj: 2.0793 loss_cls_rel: 1.7116
Epoch: [0][39000/46164] Batch_Time:  0.590      FRCNN Loss:  6.4975     RPN Loss:  1.1657
         [object] loss_cls_obj: 1.5772 loss_reg_obj: 2.0758 loss_cls_rel: 1.7037
Epoch: [0][40000/46164] Batch_Time:  0.590      FRCNN Loss:  6.4792     RPN Loss:  1.1693
         [object] loss_cls_obj: 1.5727 loss_reg_obj: 2.0721 loss_cls_rel: 1.6979
Epoch: [0][41000/46164] Batch_Time:  0.589      FRCNN Loss:  6.4576     RPN Loss:  1.1730
         [object] loss_cls_obj: 1.5673 loss_reg_obj: 2.0688 loss_cls_rel: 1.6908
Epoch: [0][42000/46164] Batch_Time:  0.589      FRCNN Loss:  6.4361     RPN Loss:  1.1766
         [object] loss_cls_obj: 1.5616 loss_reg_obj: 2.0651 loss_cls_rel: 1.6840
Epoch: [0][43000/46164] Batch_Time:  0.589      FRCNN Loss:  6.4124     RPN Loss:  1.1800
         [object] loss_cls_obj: 1.5558 loss_reg_obj: 2.0616 loss_cls_rel: 1.6760
Epoch: [0][44000/46164] Batch_Time:  0.588      FRCNN Loss:  6.3924     RPN Loss:  1.1832
         [object] loss_cls_obj: 1.5505 loss_reg_obj: 2.0582 loss_cls_rel: 1.6698
Epoch: [0][45000/46164] Batch_Time:  0.588      FRCNN Loss:  6.3749     RPN Loss:  1.1865
         [object] loss_cls_obj: 1.5456 loss_reg_obj: 2.0549 loss_cls_rel: 1.6646
Epoch: [0][46000/46164] Batch_Time:  0.588      FRCNN Loss:  6.3552     RPN Loss:  1.1895
         [object] loss_cls_obj: 1.5406 loss_reg_obj: 2.0521 loss_cls_rel: 1.6580

============ Epoch 0 ============
==========engines_v1  Testing =======
[Evaluation][500/10000][0.73s/img][avg: 76 subgraphs, max: 89 subgraphs]
         Top-50 Recall:  [Pred] 5.111%    [Phr] 5.202%    [Rel] 1.529%
         Top-100 Recall: [Pred] 6.731%    [Phr] 7.004%    [Rel] 1.985%
[Evaluation][1000/10000][0.75s/img][avg: 76 subgraphs, max: 89 subgraphs]
         Top-50 Recall:  [Pred] 5.176%    [Phr] 5.451%    [Rel] 1.457%
         Top-100 Recall: [Pred] 6.713%    [Phr] 7.390%    [Rel] 1.894%
```

Evaluation result

```
))) cat ./scripts/eval_vg_msdn.sh
#!/usr/bin/env bash

python train_FN.py --evaluate --dataset_option=normal \
--path_opt options/models/VG-MSDN.yaml --use_gt_boxes \
--pretrained_model output/trained_models/Model-VG-MSDN.h5
```

```
sungeun.kim@duchamp /home2/sungeun.kim/SceneGraph/FactorizableNet :master conda:FactorizableNet  4m 32s
        * ?
))) srun1 ./scripts/eval_vg_msdn.sh
train_FN.py:134: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader i
s unsafe. Please read https://msg.pyyaml.org/load for full details.
  options_yaml = yaml.load(handle)
train_FN.py:137: YAMLLoadWarning: calling yaml.load() without Loader=... is deprecated, as the default Loader i
s unsafe. Please read https://msg.pyyaml.org/load for full details.
  data_opts = yaml.load(f)
## args
{'MPS_iter': None,
 'clip_gradient': True,
 'dataset_option': 'normal',
 'dir_logs': None,
 'dropout': None,
 'epochs': None,
 'eval_epochs': 1,
 'evaluate': True,
 'evaluate_object': False,
 'infinite': False,
 'iter_size': 1,
 'learning_rate': None,
 'loss_weight': True,
 'model_name': None,
 'nms': -1.0,
 'optimize_MPS': False,
 'optimizer': None,
 'path_opt': 'options/models/VG-MSDN.yaml',
 'pretrained_model': 'output/trained_models/Model-VG-MSDN.h5',
 'print_freq': 1000,
 'resume': None,
 'rpn': None,
 'save_all_from': None,
 'seed': 1,
 'start_epoch': 0,
 'step_size': None,
 'triplet_nms': 0.4,
 'use_gt_boxes': True,
 'use_normal_anchors': False,
 'warm_iters': -1,
 'workers': 1}
```

```
==========engines_v1  Testing =======
[Evaluation][500/10000][0.25s/img][avg: 124 subgraphs, max: 243 subgraphs]
        Top-50 Recall:  [Pred] 44.946%  [Phr] 26.375%   [Rel] 20.967%
        Top-100 Recall: [Pred] 57.495%  [Phr] 31.303%   [Rel] 25.599%
[Evaluation][1000/10000][0.26s/img][avg: 123 subgraphs, max: 243 subgraphs]
        Top-50 Recall:  [Pred] 45.639%  [Phr] 26.291%   [Rel] 21.333%
        Top-100 Recall: [Pred] 57.746%  [Phr] 30.767%   [Rel] 25.763%
[Evaluation][1500/10000][0.25s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 46.353%  [Phr] 26.476%   [Rel] 21.766%
        Top-100 Recall: [Pred] 58.440%  [Phr] 30.795%   [Rel] 26.055%
[Evaluation][2000/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.975%  [Phr] 26.440%   [Rel] 21.311%
        Top-100 Recall: [Pred] 57.948%  [Phr] 30.819%   [Rel] 25.564%
[Evaluation][2500/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.709%  [Phr] 26.018%   [Rel] 21.102%
        Top-100 Recall: [Pred] 57.659%  [Phr] 30.450%   [Rel] 25.258%
[Evaluation][3000/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.588%  [Phr] 26.106%   [Rel] 21.096%
        Top-100 Recall: [Pred] 57.458%  [Phr] 30.543%   [Rel] 25.187%
[Evaluation][3500/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.640%  [Phr] 25.872%   [Rel] 21.006%
        Top-100 Recall: [Pred] 57.368%  [Phr] 30.258%   [Rel] 25.015%
[Evaluation][4000/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.787%  [Phr] 25.732%   [Rel] 20.972%
        Top-100 Recall: [Pred] 57.498%  [Phr] 30.153%   [Rel] 24.949%
[Evaluation][4500/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.531%  [Phr] 25.765%   [Rel] 20.923%
        Top-100 Recall: [Pred] 57.234%  [Phr] 30.269%   [Rel] 24.974%
[Evaluation][5000/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.526%  [Phr] 25.742%   [Rel] 20.871%
        Top-100 Recall: [Pred] 57.215%  [Phr] 30.297%   [Rel] 24.913%
[Evaluation][5500/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
```

```
[Evaluation][6500/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.634%  [Phr] 25.997%   [Rel] 21.070%
        Top-100 Recall: [Pred] 57.457%  [Phr] 30.604%   [Rel] 25.207%
[Evaluation][7000/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.544%  [Phr] 25.914%   [Rel] 21.011%
        Top-100 Recall: [Pred] 57.387%  [Phr] 30.497%   [Rel] 25.174%
[Evaluation][7500/10000][0.26s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.656%  [Phr] 26.036%   [Rel] 21.069%
        Top-100 Recall: [Pred] 57.433%  [Phr] 30.579%   [Rel] 25.206%
[Evaluation][8000/10000][0.27s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.648%  [Phr] 26.022%   [Rel] 21.102%
        Top-100 Recall: [Pred] 57.392%  [Phr] 30.570%   [Rel] 25.253%
[Evaluation][8500/10000][0.27s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.632%  [Phr] 26.106%   [Rel] 21.129%
        Top-100 Recall: [Pred] 57.359%  [Phr] 30.638%   [Rel] 25.293%
[Evaluation][9000/10000][0.27s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.625%  [Phr] 26.063%   [Rel] 21.099%
        Top-100 Recall: [Pred] 57.364%  [Phr] 30.599%   [Rel] 25.279%
[Evaluation][9500/10000][0.27s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.548%  [Phr] 25.979%   [Rel] 21.061%
        Top-100 Recall: [Pred] 57.324%  [Phr] 30.536%   [Rel] 25.265%
[Evaluation][10000/10000][0.27s/img][avg: 124 subgraphs, max: 363 subgraphs]
        Top-50 Recall:  [Pred] 45.575%  [Phr] 25.921%   [Rel] 21.047%
        Top-100 Recall: [Pred] 57.388%  [Phr] 30.521%   [Rel] 25.270%

====== Done Testing ====
======= Testing Result =======
Top-50 Recall   [Pred]: 45.575% [Phr]: 25.921%  [Rel]: 21.047%
Top-100 Recall  [Pred]: 57.388% [Phr]: 30.521%  [Rel]: 25.270%
=========== Done ===========
```