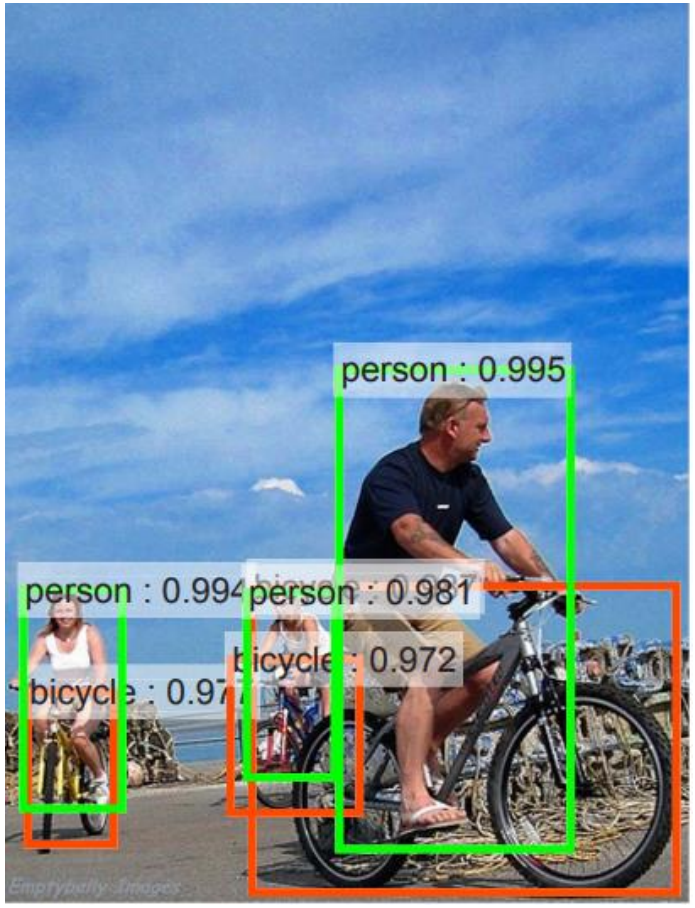
VSGNet: Spatial Attention Network for Detecting Human Object Interactions Using Graph Convolutions

CVPR 2020

Youngwook Kim

Task: Human-object Interaction Detection



<Object Detection>

v.s.

<Interaction Detection>

Kick, Drink, Carry, Hold, Hit, Eat, ….

Existing Methods & Contribution of the Paper

1. Extract human features & objects features
   using object detection network.

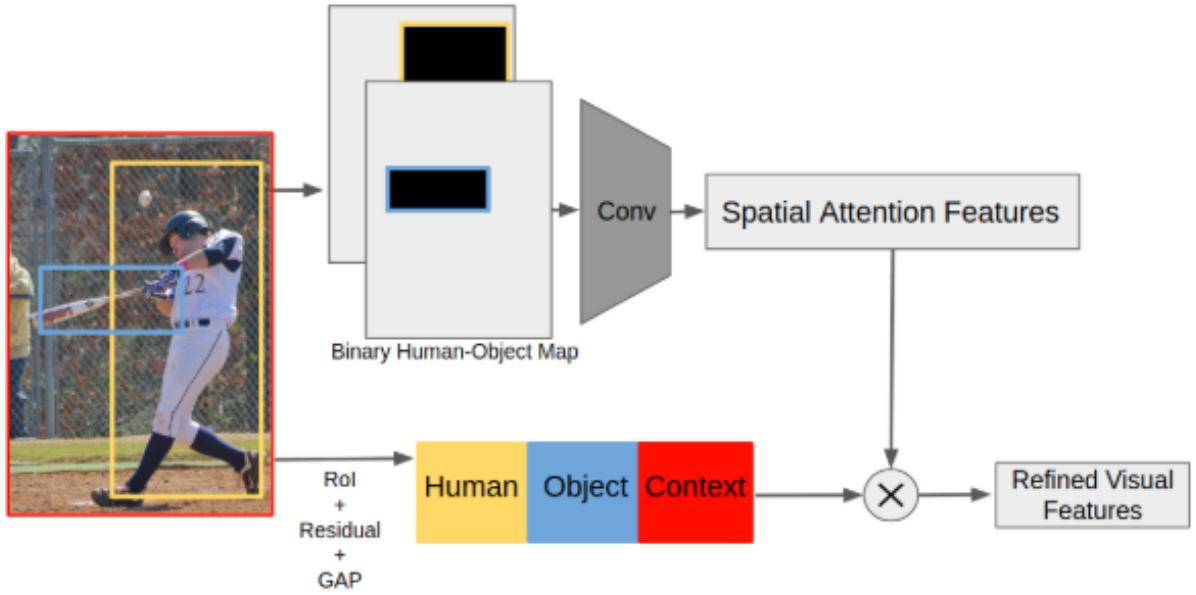
2. Pair these features exhaustively.

3. Fed feature pairs into a multi-branch deep neural network
   to detect the relationship between humans and objects.

⇒ It does not explicitly utilize the interaction information
   or the spatial relations between pairs.

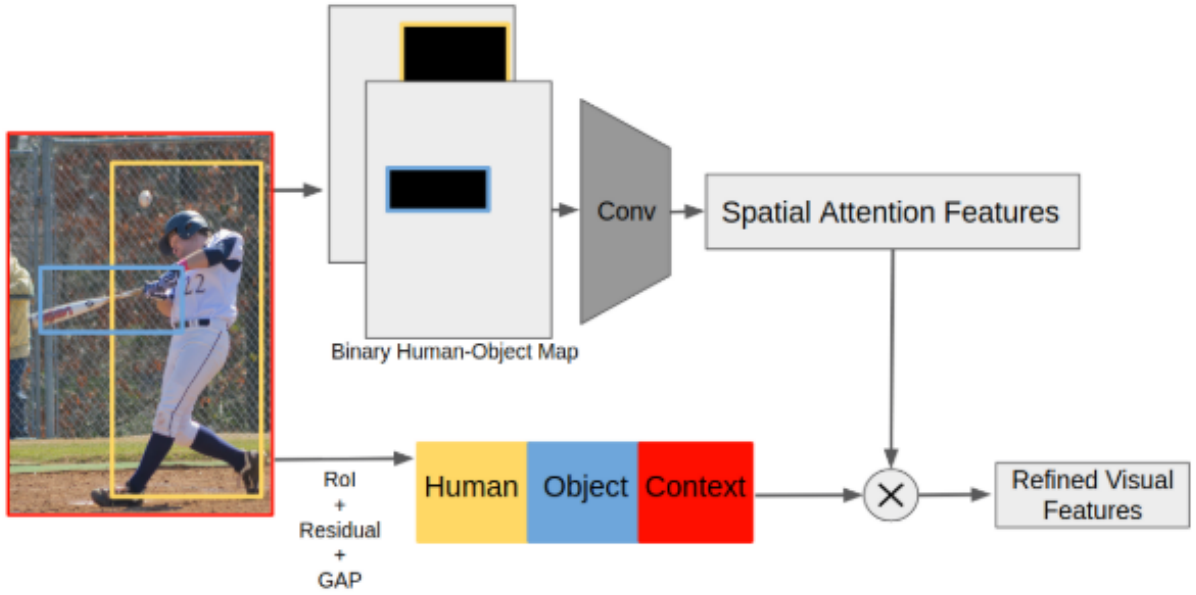
⇒ Graph Convolutional Branch, Spatial Attention Branch

Spatial Attention Branch



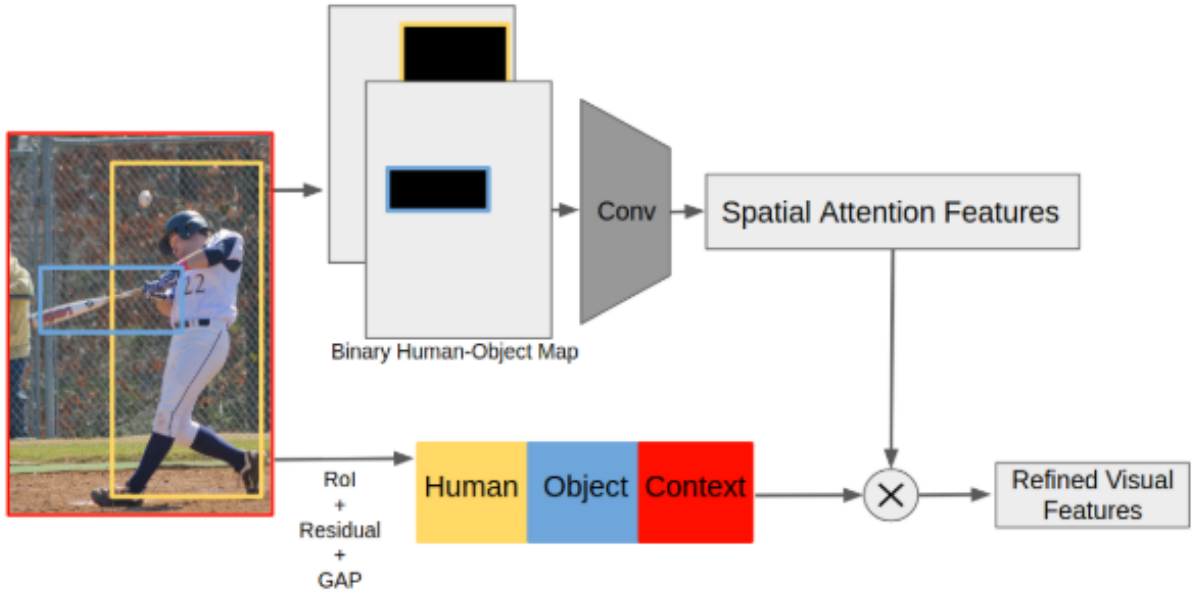
1. Obtain Human bounding boxes and object bounding boxes from a pre-trained object detector.

Spatial Attention Branch



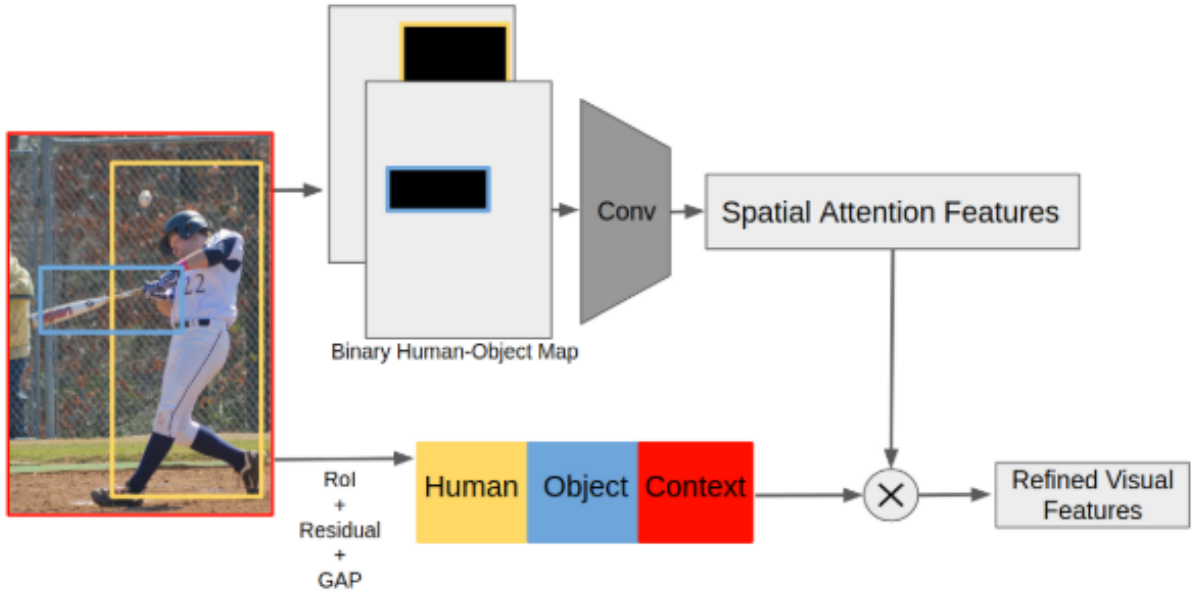
2. Human, object, and context visual features are extracted from the image using RoI pooling.

Spatial Attention Branch



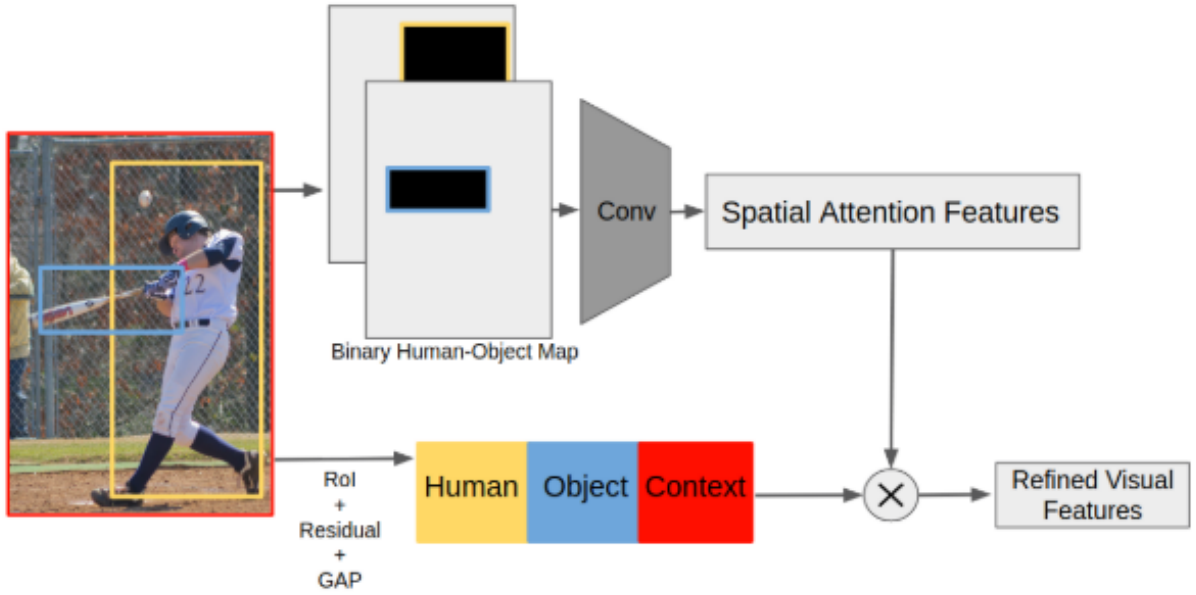
3. Using binary maps of human and object locations, spatial attention features are extracted using convolutions.

Spatial Attention Branch



4. Attention features are multiplied to refine the visual features by amplifying the pairs with high spatial correlation.

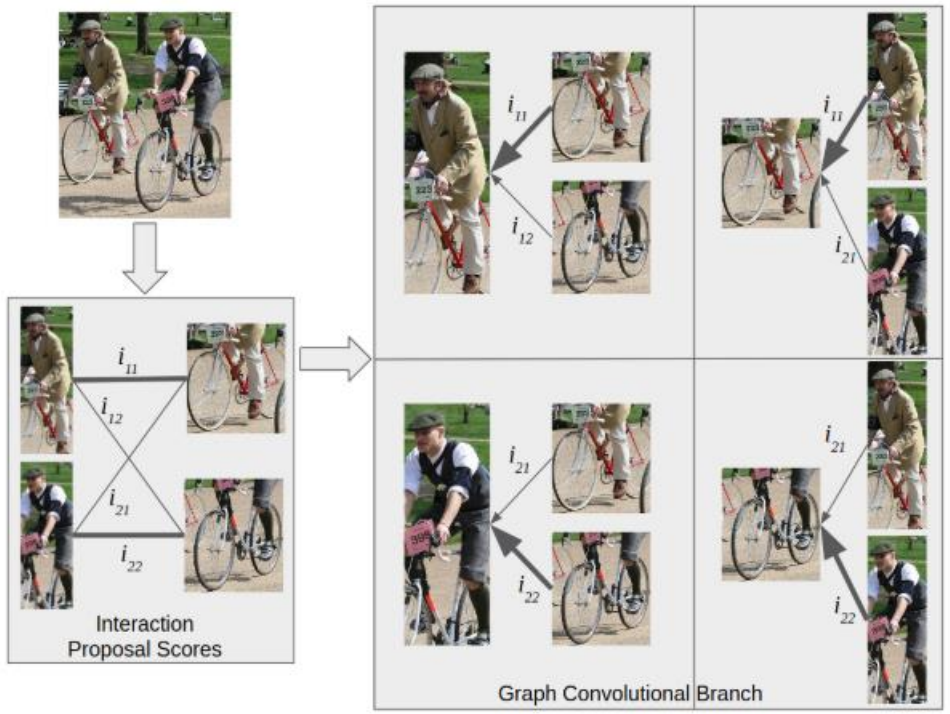
Spatial Attention Branch



5. The refined feature vector is then used to predict the interaction proposal score of human-object pair *ho*.

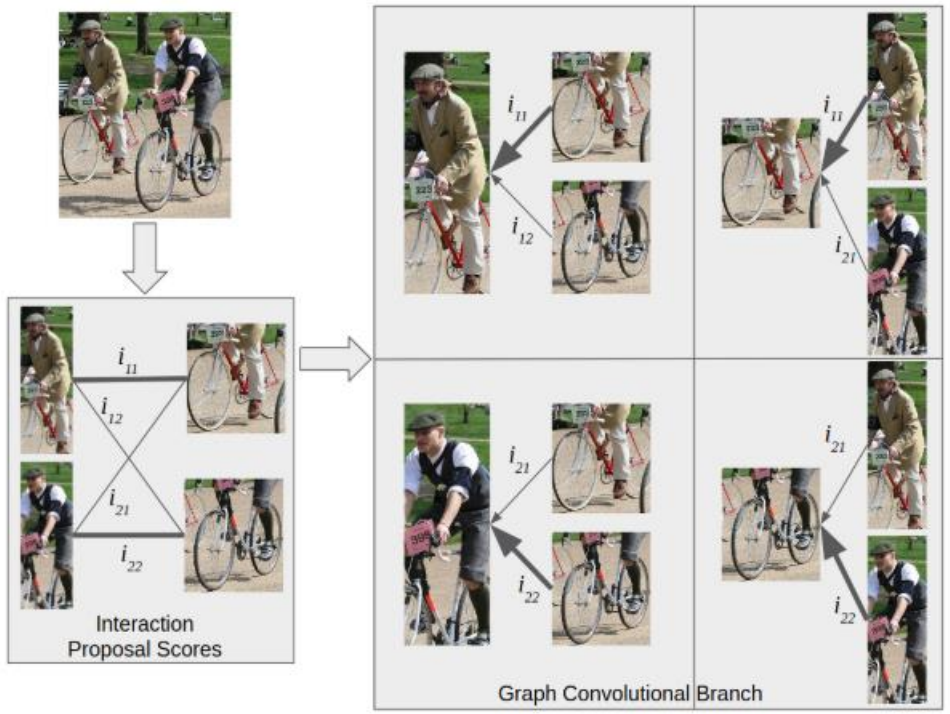
$$i_{ho} = \sigma(\mathbf{W}_{IP}(\mathbf{f}_{ho}^{Ref}))$$

Graph Convolutional Branch



1. Define the humans and objects as nodes
   and only connecting edges between human-object pairs.

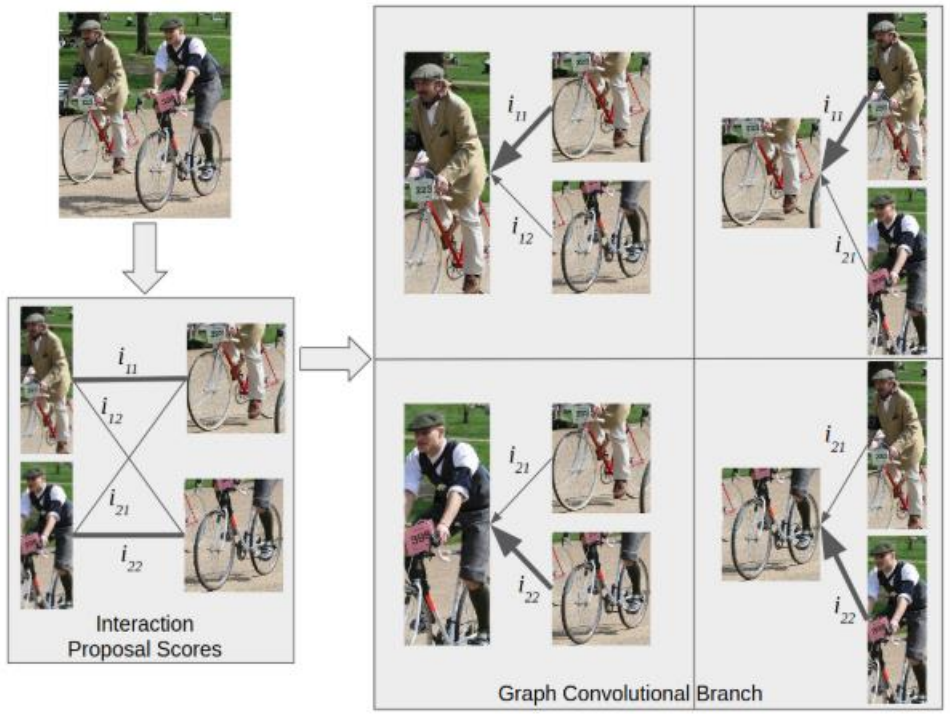
Graph Convolutional Branch



2. Previous works defined the adjacency as visual similarity.
   In this work, however, define adjacency values between h and o pair as interaction proposal score

$$\alpha_{ho} = \alpha_{oh} = i_{ho}$$

Graph Convolutional Branch



3. Traverse and update the nodes in the graph using their edges.

$$\mathbf{f}_h' = \mathbf{f}_h + \sum_{o=1}^{O} \alpha_{ho} \mathbf{W}_{oh}(\mathbf{f}_o)$$

$$\mathbf{f}_o' = \mathbf{f}_o + \sum_{h=1}^{H} \alpha_{oh} \mathbf{W}_{ho}(\mathbf{f}_h)$$

Model Architecture

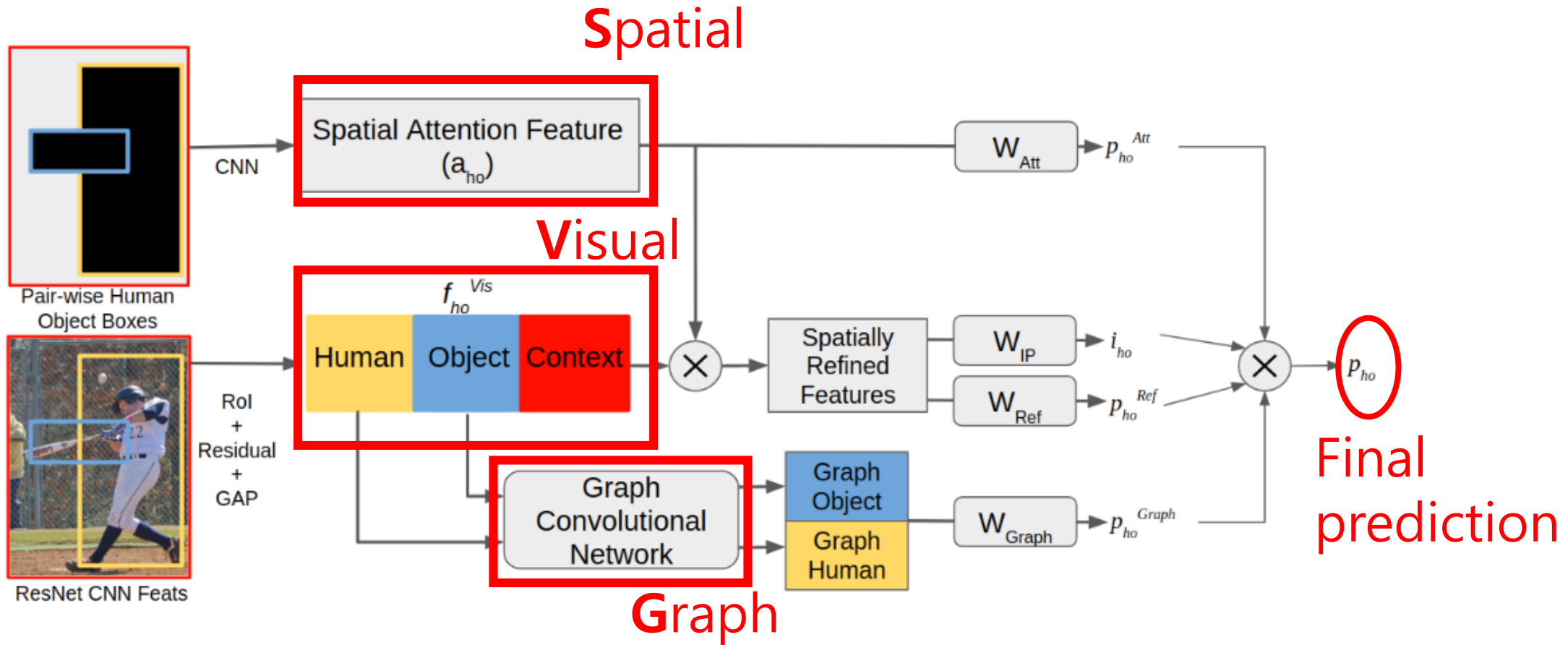


Figure 2. Model Architecture. Rounded rectangles are operations, sharp rectangles are extracted features and ⊗ is element-wise multiplication. The model consists of three main b…ject and context features. Spatial Attention branch refines the visual features by utilizing the s…ir. Graph Convolutional branch extracts interaction features by considering humans/objects as n… class probabilities from each branch and the interaction proposal score are multiplied together to aggregate the final prediction. These operations are repeated for every human-object pair.

➢ VSGNet

Experimental Results

<Comparison with SOTA>

V-COCO	mAP(Sc 1)	mAP(Sc 2)
InteractNet [7]	40.0	47.98
Kolesnikov et al. [13]	41.0	-
Graph기반-> GPNN [21]	44.0	-
iCAN [4]	45.3	52.4
Li et al. [15]	47.8	-
VSGNet	**51.76**	**57.03**

HICO-DET (mAP)	Full	Rare	Non-Rare
HO-RCNN [1]	7.81	5.37	8.54
InteractNet [7]	9.94	7.16	10.77
Graph기반-> GPNN [21]	13.11	9.34	9.34
iCAN [4]	14.84	10.45	16.15
Li et al. [15]	17.03	13.42	18.11
VSGNet	**19.80**	**16.05**	**20.91**

<Ablation Study>

Branches	mAP(Sc 1)	mAP(Sc 2)
Visual (Base)	47.3	52.15
Visual+Graph	48.19	53.12
Visual+Spatial	50.33	55.32
Visual+Spatial+Graph(VSG)	51.76	57.03

Table 4. Analysis of the branches. Our base model consists of only the Visual branch. We add the graph branch and the spatial attention branch to this base model separately to analyze their performances. Individually, both branches improve the performance upon the base model. Visual+Spatial model beats the state of the art results and all three branches combined adds another 1.5 mAP.

Experimental Results

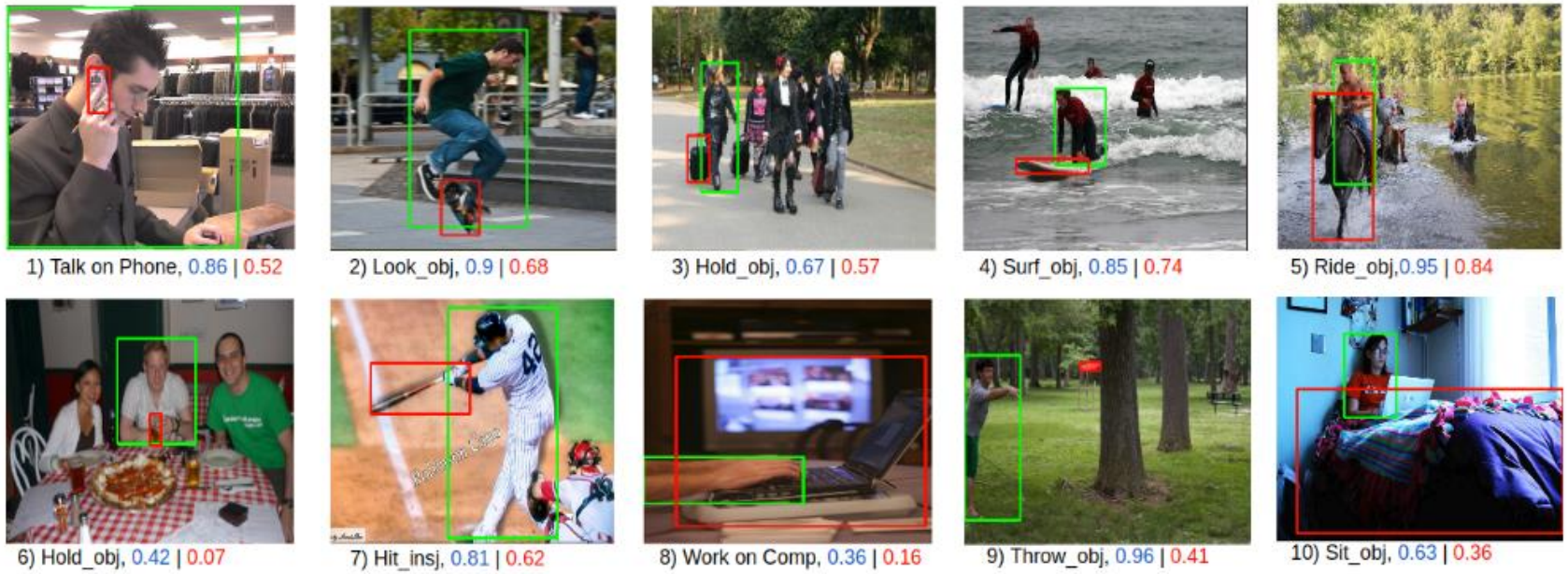
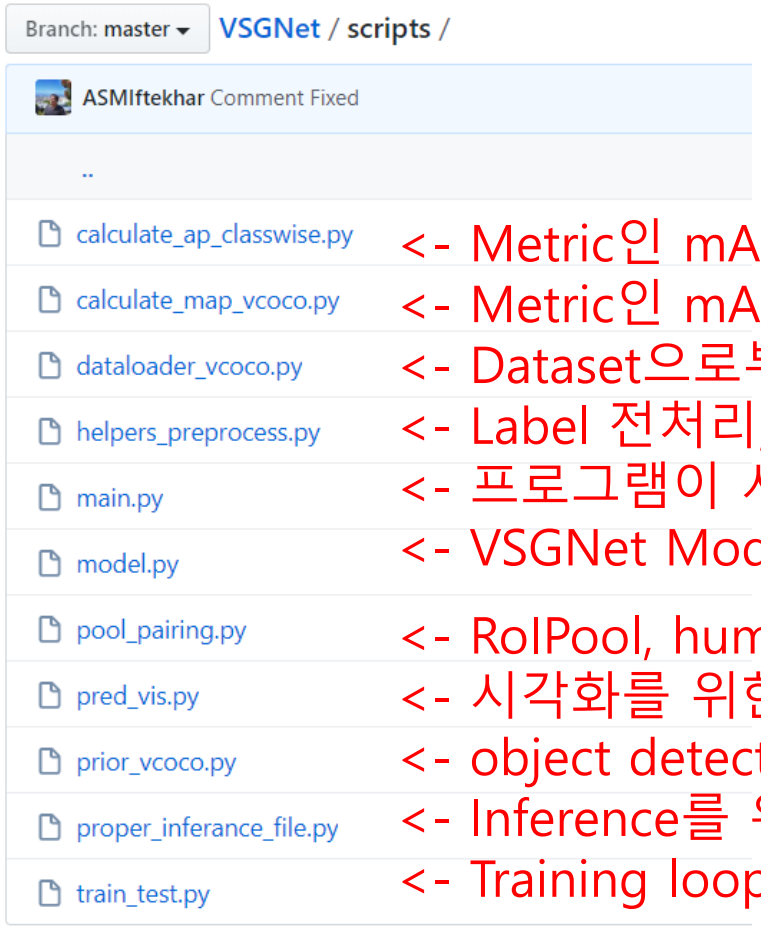


Figure 5. Qualitative results. Red values show the confidences for the base model (Visual only) and blue values are the results for the VSGNet. The prediction results and the correct action labels are shown for the human-object pair visualized with the bounding boxes.

Paper Reproducing

1. 저자가 Github에 공개한 코드를 기본적으로 참조하였습니다.
   (https://github.com/ASMIftekhar/VSGNet)

코드의 구조



<- Metric인 mAP를 계산하는 코드
<- Metric인 mAP를 계산하는 코드 (API 사용)
<- Dataset으로부터 Image, Label Minibatch 생성하는 코드
<- Label 전처리, IoU 구하는 등 함수를 담아둔 코드
<- 프로그램이 시작하는 Main 코드
<- VSGNet Model이 구현되어 있는 코드
<- RoIPool, human-object feature paring이 구현되어 있는 코드
<- 시각화를 위한 코드 (사용하지 않음)
<- object detector 로부터의 prior를 가져오는 코드
<- Inference를 위한 코드 (사용하지 않음)
<- Training loop와 Test를 통해 metric를 출력하는 코드

Paper Reproducing

2. 논문 내용 및 Code Implementation을 공부하기 위한 목적으로, 공개코드를 모방하여 코딩을 해보았습니다. (4~5월)

calculate_ap_classwise.py	한 달 전	2.93 kB
calculate_map_vcoco.py	2달 전	2.25 kB
dataloader_vcoco.py	2달 전	3.57 kB
helpers_preprocess.py	2달 전	12.7 kB
main.py	2달 전	5.99 kB
main_ywkim.py <- main 함수 reimplementation	한 달 전	2.89 kB
model.py	2달 전	11.3 kB
model_ywkim.py <- model 함수 reimplementation	한 달 전	9.75 kB
people.jpg	3년 전	1.92 MB
pool_pairing.py	2달 전	4.23 kB
pred_vis.py	2달 전	4.37 kB
prior_vcoco.py	2달 전	679 B
proper_inferance_file.py	2달 전	3.56 kB
train_test.py	2달 전	16.1 kB
train_test_ywkim.py <- train_test 함수 reimplementation	한 달 전	11.9 kB

Paper Reproducing

3. main_ywkim.py 코드의 구조를 자세히 설명하겠습니다.

jupyter main_ywkim.py 2020.05.01

File Edit View Language

```
1  import torch
2  import torchvision
3  import json
4  import torch.nn as nn
5  import json
6  import sys
7  import warnings
8  from torch.utils.data import Dataset, DataLoader
9  from dataloader_vcoco import Rescale,ToTensor,vcoco_Dataset, vcoco_collate
10 from torchvision import transforms
11 import numpy as np
12 import random
13 import os
14 from tqdm import tqdm
15 import torch.optim as optim
16
17 from model_ywkim import VSGNet
18 from train_test_ywkim import train_test
19 warnings.filterwarnings("ignore")
20
21 device= torch.device('cuda')
22
23 num_epochs = 50
24 batch_size=8
25
26 seed=10
27 torch.manual_seed(seed)
28 torch.backends.cudnn.deterministic = True
29 torch.backends.cudnn.benchmark = False
30 np.random.seed(seed)
31 random.seed(seed)
32 os.environ['PYTHONHASHSEED'] = str(seed)
33 torch.cuda.manual_seed(seed)
34 torch.cuda.manual_seed_all(seed)
35 def _init_fn(worker_id):
36     np.random.seed(int(seed))
37
```

Function, Library import

Hyperparameter 설정

Seed 설정

Paper Reproducing

```
# with open('../infos/directory.json') as fp: all_data_dir=json.load(fp)
all_data_dir = '../All_data_vcoco/'

annotation_train=all_data_dir+'Annotations_vcoco/train_annotations.json'
image_dir_train=all_data_dir+'Data_vcoco/train2014/'

annotation_val=all_data_dir+'Annotations_vcoco/val_annotations.json'
image_dir_val=all_data_dir+'Data_vcoco/train2014/'

annotation_test=all_data_dir+'Annotations_vcoco/test_annotations.json'
image_dir_test=all_data_dir+'Data_vcoco/val2014/'

vcoco_train=vcoco_Dataset(annotation_train,image_dir_train,transform=transforms.Compose([Rescale((400,400)),ToTensor()]))
vcoco_val=vcoco_Dataset(annotation_val,image_dir_val,transform=transforms.Compose([Rescale((400,400)),ToTensor()]))
vcoco_test=vcoco_Dataset(annotation_test,image_dir_test,transform=transforms.Compose([Rescale((400,400)),ToTensor()]))


#import pdb;pdb.set_trace()
dataloader_train = DataLoader(vcoco_train, batch_size,
                        shuffle=True,collate_fn=vcoco_collate,num_workers=8,worker_init_fn=_init_fn)#num_workers=batch_size
dataloader_val = DataLoader(vcoco_val, batch_size,
                        shuffle=True,collate_fn=vcoco_collate,num_workers=8,worker_init_fn=_init_fn)#num_workers=batch_size
dataloader_test = DataLoader(vcoco_test, batch_size,
                        shuffle=False,collate_fn=vcoco_collate,num_workers=8, worker_init_fn=_init_fn)#num_workers=batch_size
dataloader={'train':dataloader_train,'val':dataloader_val,'test':dataloader_test}


model = VSGNet()

for name, p in model.named_parameters():
    if name.split('.')[0] == 'Conv_pretrain':
        p.requires_grad = False

```

V-COCO Dataset 가져오고 Batch 만들기

Model 가져오고 앞단의 convolutional layer freeze

Paper Reproducing

```
73 optim1 = optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight_decay=0.0001)
74
75
76 lambda1 = lambda epoch: 1.0 if epoch < 10 else (10 if epoch < 28 else 1)
77 lambda2 = lambda epoch: 1
78 lambda3 = lambda epoch: 1
79 scheduler=optim.lr_scheduler.LambdaLR(optim1,lambda1)
80
81 model = nn.DataParallel(model)
82 model.to(device)
83
84 epoch = 0
85 mean_best = 0
86
87 train_test(model, optim1, scheduler, dataloader, num_epochs, batch_size, epoch, mean_best)
```

SGD로 Model Train

Paper Reproducing

4. 이번에는 model_ywkim.py 코드의 구조를 자세히 설명하겠습니다.

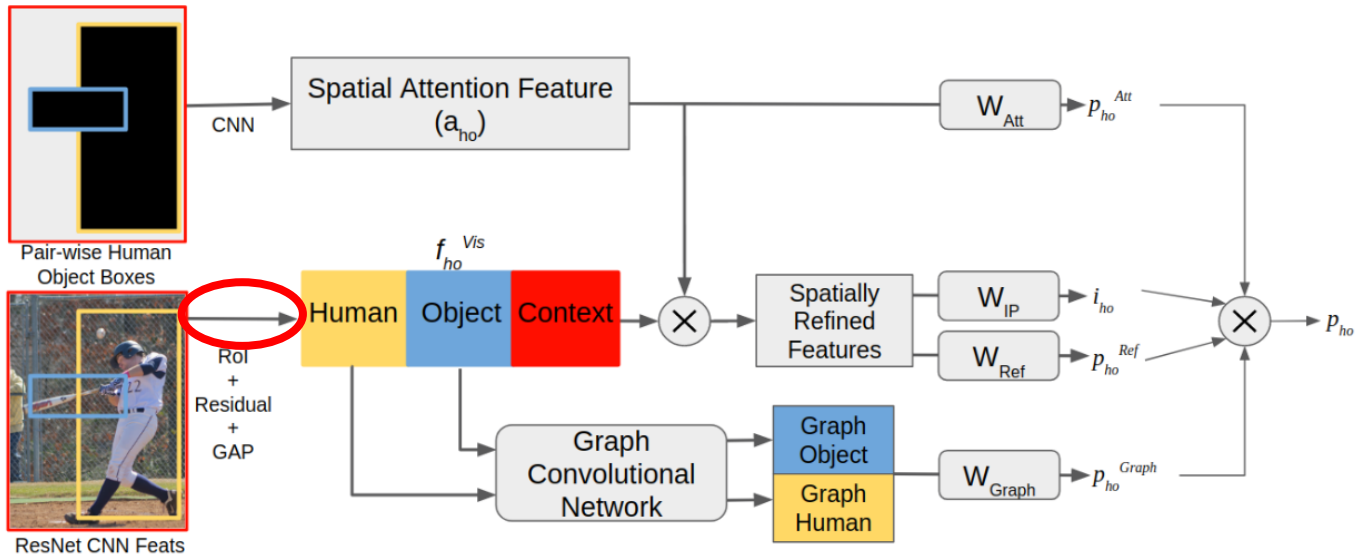
```
108     def forward(self, x, pairs_info, pairs_info_augmented, image_id, flag_, phase):
109
110         out1 = self.Conv_pretrain(x)
111
112         rois_people, rois_objects, spatial_locs, union_box = ROI.get_pool_loc(out1, image_id, flag_, size=pool_size, spatial_scale=25,
    batch_size=len(pairs_info))
113
114         ### Defining The Pooling Operations #######
115         x,y=out1.size()[2],out1.size()[3]
116         hum_pool=nn.AvgPool2d(pool_size,padding=0,stride=(1,1))
117         obj_pool=nn.AvgPool2d(pool_size,padding=0,stride=(1,1))
118         context_pool=nn.AvgPool2d((x,y),padding=0,stride=(1,1))
119         ##################################################
```

<- pretrained ResNet152 model로 Image에서 CNN Featuremap을 얻음

RoIPooling하는 함수를 pre-define
(Average Pooling 사용)

Paper Reproducing

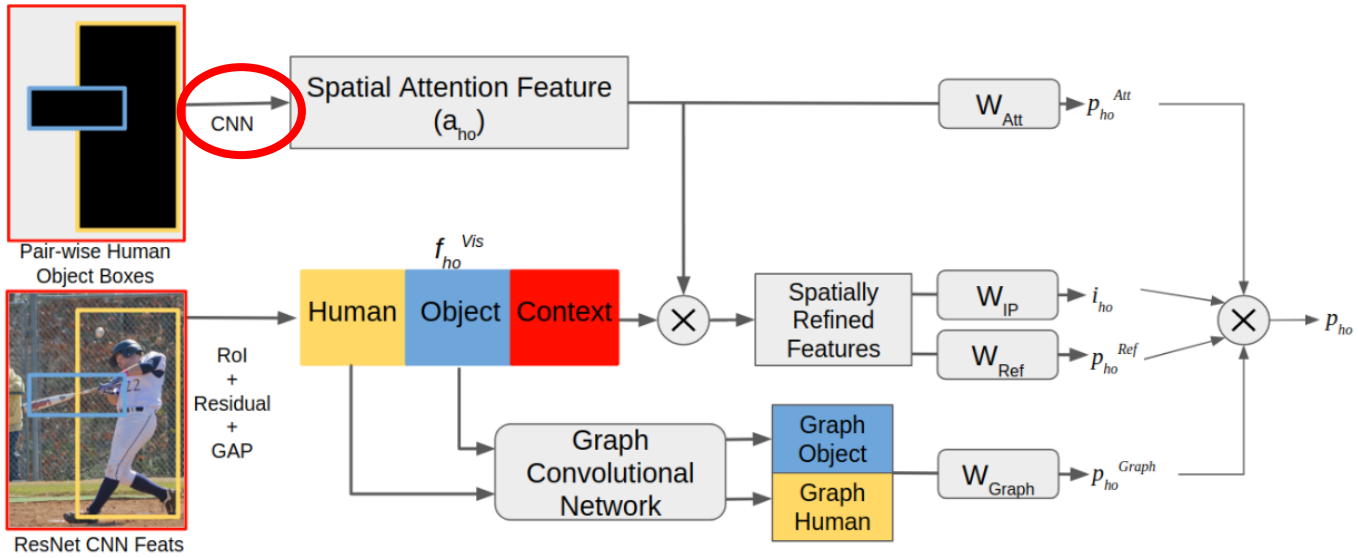
```
123         ### Human###
124         residual_people=rois_people
125         res_people=self.Conv_people(rois_people)+residual_people
126         res_av_people=hum_pool(res_people)
127         out2_people=self.flat(res_av_people)
128         ###########
129
130
131         ##Objects##
132         residual_objects=rois_objects
133         res_objects=self.Conv_objects(rois_objects)+residual_objects
134         res_av_objects=obj_pool(res_objects)
135         out2_objects=self.flat(res_av_objects)
136         #############
137
138
139         #### Context ######
140         residual_context=out1
141         res_context=self.Conv_context(out1)+residual_context
142         res_av_context=context_pool(res_context)
143         out2_context=self.flat(res_av_context)
144         ################
```



Visual branch에 해당. Human feature/ object feature/ context feature 얻는 부분

Paper Reproducing

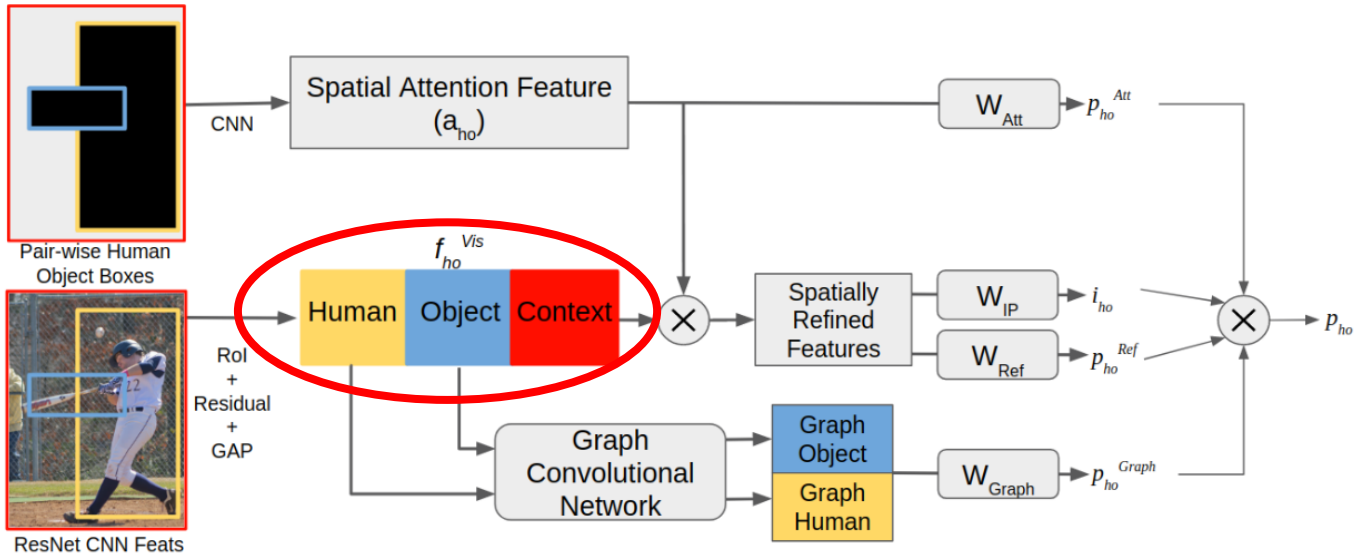
```
146
147
        ## Attention features ##
        a_ho = self.W_Spat(self.flat(self.Conv_spatial(union_box)))
```



Spatial Branch에 해당. Binary map을 CNN에 통과시켜 Spatial Attention Feature a_ho를 얻는 부분

Paper Reproducing

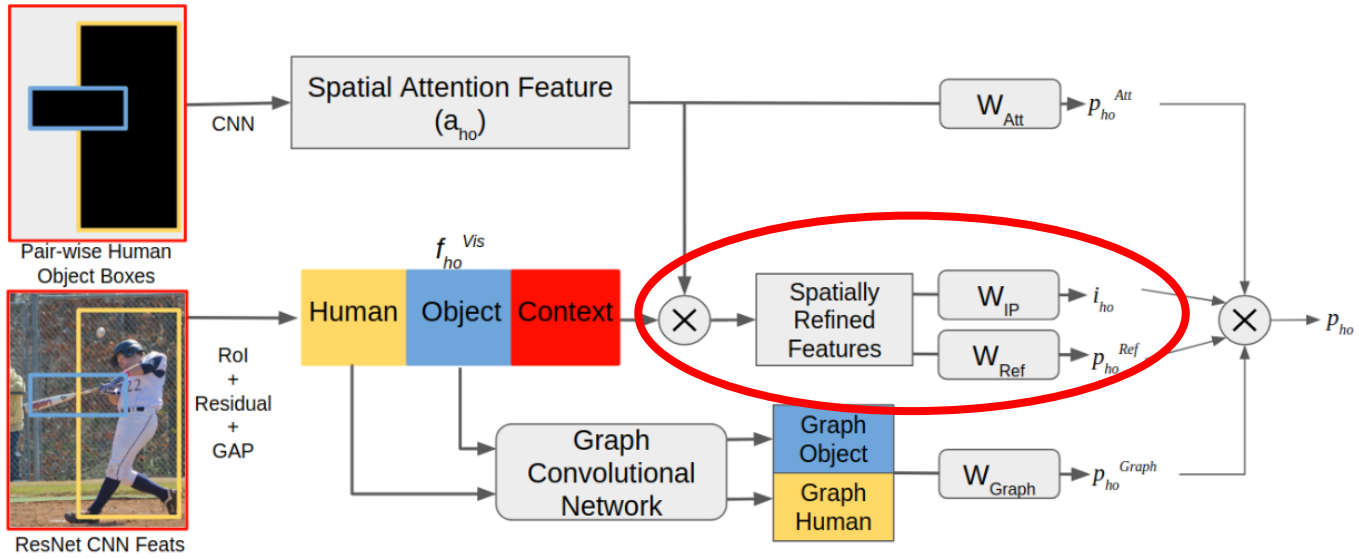
```
149     ### Making Essential Pairing##########
150     pairs, people, objects_only = ROI.pairing(out2_people, out2_objects, out2_context, spatial_locs, pairs_info)
151     ######################################
```



Human feature와 object feature 를 하나씩 pairing하여 concatenated feature를 얻는 부분
예를 들어 이미지 내에서 사람이 세명, 물체가 네개 검출되었다면 총 12개의 concatenated feature를 생성함

Paper Reproducing

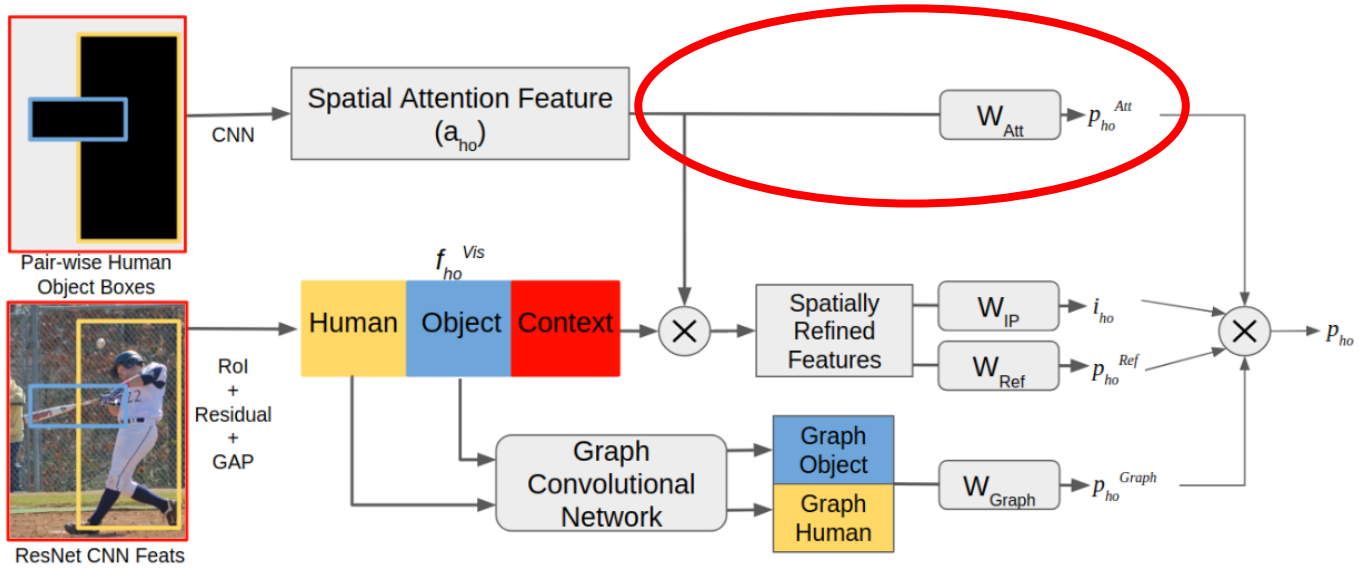
```
153         ### Interaction Probability ########
154         f_Vis = self.W_vis(pairs)
155         f_Ref = f_Vis * a_ho
156         i_ho = self.W_IP(f_Ref)
157         interaction_prob = self.sigmoid(i_ho)
158         p_Ref = self.W_Ref(f_Ref)
```



Concatenated feature에 Spatial attention featur를 곱해 Spatially refined feature를 얻고,
이를 기반으로 interaction proposal score i_ho와 prediction score p_Ref를 얻는 부분

Paper Reproducing

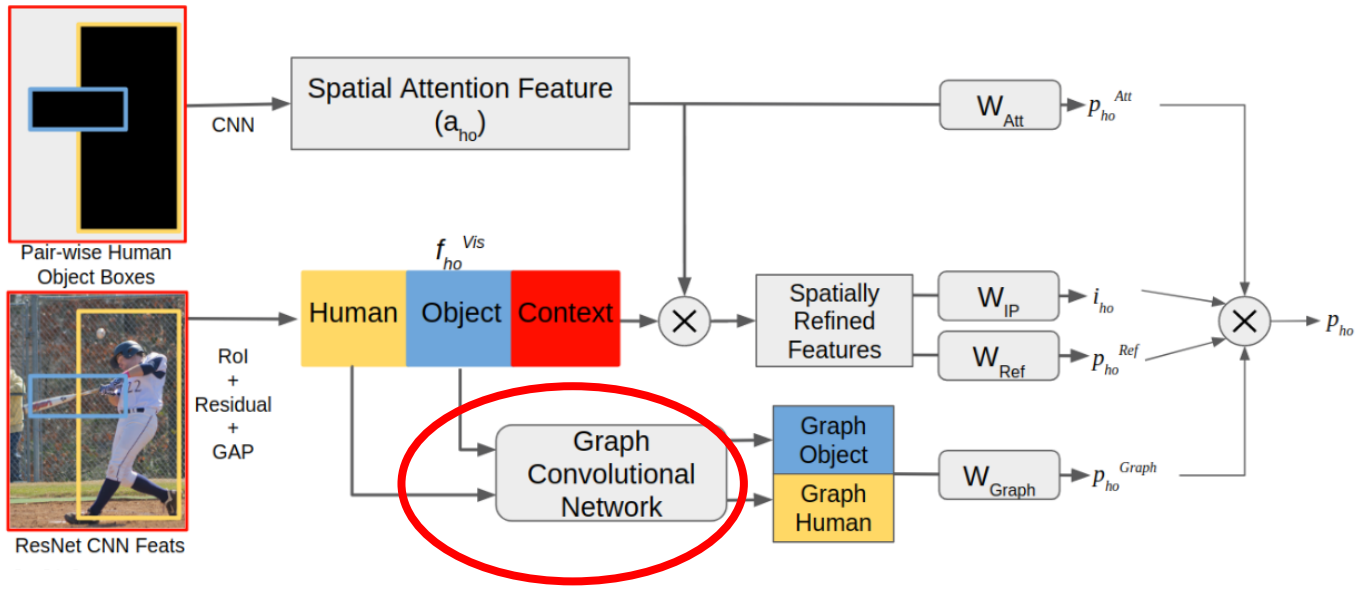
```
160         ### Prediction from attention features
161
162         p_Att = self.W_Att(a_ho)
```



Spatial attention feature로부터 prediction score p_Att를 얻는 부분

Paper Reproducing

```
165         ### Graph model base structure
166         people_t = people
167         objects_only = objects_only
168         combine_g = []
169         people_f = []
170         objects_f = []
171         pairs_f = []
172         start_p = 0
173         start_o = 0
174         start_c = 0
175
176         for batch_num, l in enumerate(pairs_info):
177
178             ### Slicing ###
179             people_this_batch = people_t[start_p : start_p + int(l[0])]
180             num_peo = len(people_this_batch)
181
182             objects_this_batch = objects_only[start_o : start_o + int(l[1]
183             # because first index means no object
184             no_objects_this_batch = objects_only[start_o : start_o + int(l
185             num_obj = len(objects_this_batch)
186
187             interaction_prob_this_batch = interaction_prob[start_c : start_
188                                                            int(l[0]) * int
189
190             if num_obj == 0:
191                 people_this_batch_r = people_this_batch # r means refine
192                 objects_this_batch_r = no_objects_this_batch.view([1, 1024])
193             else:
194                 peo_to_obj_this_batch = torch.stack([torch.cat((i, j)) for ind_p, i in enumer
    enumerate(objects_this_batch)])
195                 obj_to_peo_this_batch = torch.stack([torch.cat((i, j)) for ind_p, i in enumer
    enumerate(people_this_batch)])
196
```

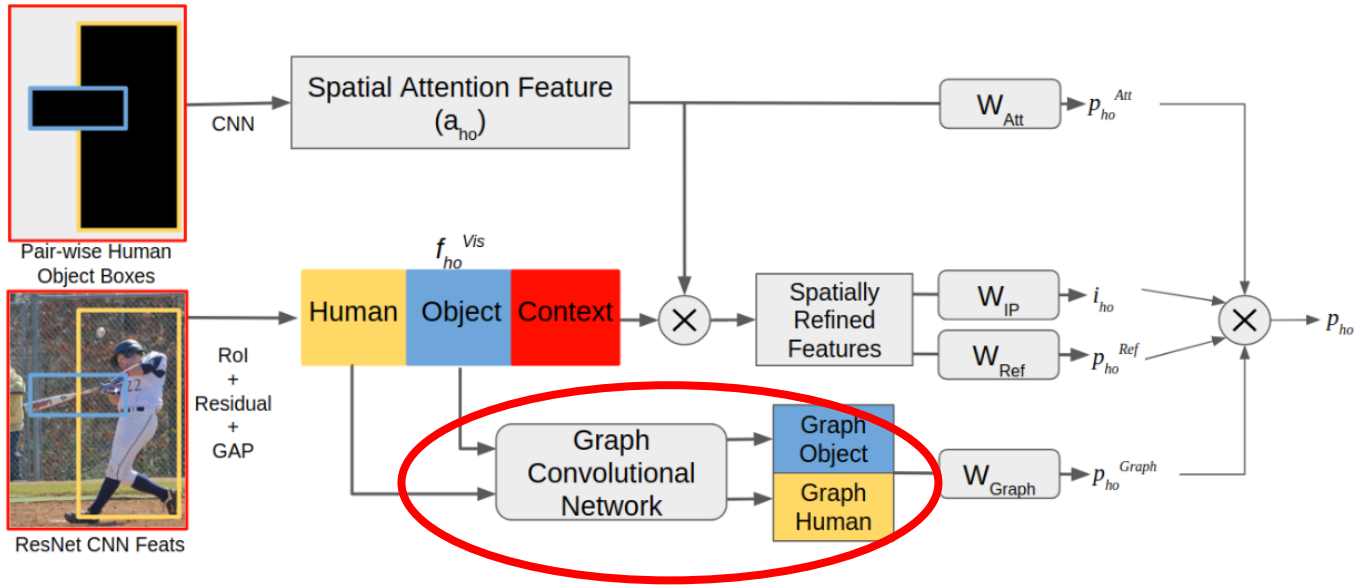


Graph의 Node인 Human, object 의 node feature를 가져오는 부분

Paper Reproducing

```
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
```

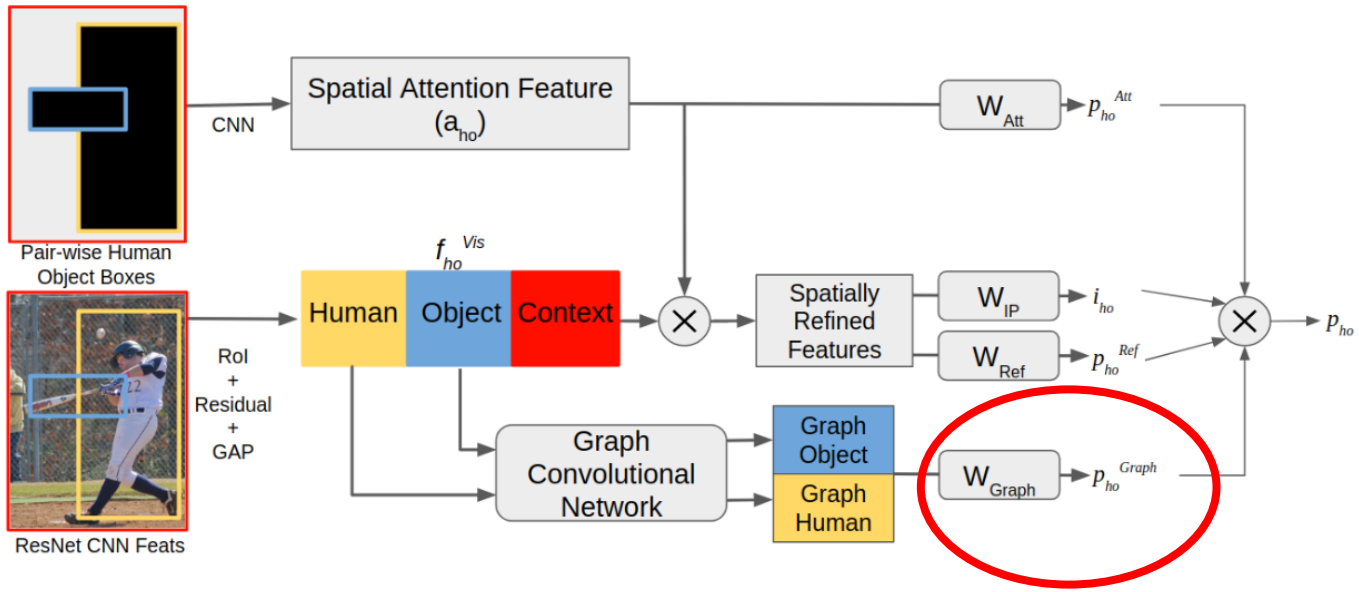
```
###################

    ## Adjacency ###
    adj_l = []
    adj_po = torch.zeros([num_peo, num_obj]).cuda()
    adj_op = torch.zeros([num_obj, num_peo]).cuda()

    for index_probs, probs in enumerate(interaction_prob_this_
        if index_probs % (num_obj + 1) != 0:
            adj_l.append(probs)

    adj_po = torch.cat(adj_l).view(len(adj_l),1)# no gradient
    adj_op = adj_po

    ### Finding out Refined features ###

    people_this_batch_r = people_this_batch + torch.mm(adj_po.view([num_peo,num_obj]), self.W_oh(objects_this_batch))

    objects_this_batch_r = objects_this_batch + torch.mm(adj_op.view([num_peo, num_obj]).t(), self.W_ho(people_this_batch))
    objects_this_batch_r = torch.cat((no_objects_this_batch.view([1, 1024]),₩
                                      objects_this_batch_r))
```



Interaction proposal score를 기반으로 graph의 adjacency를 구성하고,
Graph convolution을 수행하는 부분

Paper Reproducing

```
        ### Reconstructing ###
        people_f.append(people_this_batch_r)
        people_t_f = people_this_batch_r
        objects_f.append(objects_this_batch_r)
        objects_t_f = objects_this_batch_r

        pairs_f.append(torch.stack([torch.cat((i, j)) for ind_p, i in
                                    for ind_o, j in enumerat

        ## loop increment for next batch
        start_p += int(l[0])
        start_o += int(l[1])
        start_c += int(i[0]) * int(i[1])

    people_graph = torch.cat(people_f)
    objects_graph = torch.cat(objects_f)
    pairs_graph = torch.cat(pairs_f)

    p_Graph = self.W_graph(pairs_graph)
```



Human-object pair를 다시 구성한 후, prediction score p_Graph를 얻는 부분

Paper Reproducing Result

Branches	mAP(Sc 1)	mAP(Sc 2)
Visual (Base)	47.3	52.15
Visual+Graph	48.19	53.12
Visual+Spatial	50.33	55.32
Visual+Spatial+Graph(VSG)	51.76	57.03

Table 4. Analysis of the branches. Our base model consists of only the Visual branch. We add the graph branch and the spatial attention branch to this base model separately to analyze their performances. Individually, both branches improve the performance upon the base model. Visual+Spatial model beats the state of the art results and all three branches combined adds another 1.5 mAP.

Reproducing 결과 (Sc 1)
46.75
52.37
-
54.57

<논문에서 Report된 V-COCO dataset에서의 성능 지표>

cf) Reproducing했을 때는 공식 API를 사용하여 성능 지표를 낸 것이 아니어서 이로 인한 오차가 있을 수 있습니다.

Code 주소 : https://github.com/ascii1203/VSGNet


Thank
you!